厦門大學

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　王伟龙

学　　号　22920192204287

实验时间　2021 年 6 月 5 日

1、 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。

1 实验环境

Windows10

2 实验结果

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

```c
#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
};

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;
}
```

```c
//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;
            return -1;
        } else if (ret == 0) {
            struct addrinfo *n;
```

```
            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("1: %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) {
        readn(fd, (void *)&type, 1);
        log_message("Method AUTH %hhX", type);
        if (type == auth_type) {
```

```
297         if (type == auth_type) {
298             supported = 1;
299         }
300     }
301     if (supported == 0) {
302         socks5_auth_notsupported(fd);
303         app_thread_exit(1, fd);
304     }
305     int ret = 0;
306     switch (auth_type) {
307     case NOAUTH:
308         ret = socks5_auth_noauth(fd);
309         break;
310     case USERPASS:
311         ret = socks5_auth_userpass(fd);
312         break;
313     }
314     if (ret == 0) {
315         return;
316     } else {
317         app_thread_exit(1, fd);
318     }
319 }
320
321 //读取命令
322 int socks5_command(int fd)
323 {
324     char command[4];
325     readn(fd, (void *)command, ARRAY_SIZE(command));
326     log_message("Command %hhX %hhX %hhX %hhX", command[0], command[1],
327             command[2], command[3]);
328     return command[3];
329 }
330
331 //读取接口
332 unsigned short int socks_read_port(int fd)
333 {
334     unsigned short int p;
335     readn(fd, (void *)&p, sizeof(p));
336     log_message("Port %hu", ntohs(p));
337     return p;
338 }
339
340 //读取ip
341 char *socks_ip_read(int fd)
342 {
343     char *ip = (char *)malloc(sizeof(char) * IPSIZE);
344     readn(fd, (void *)ip, IPSIZE);
345     log_message("IP %hhu.%hhu.%hhu.%hhu", ip[0], ip[1], ip[2], ip[3]);
346     return ip;
347 }
348
349 void socks5_ip_send_response(int fd, char *ip, unsigned short int port)
350 {
351     char response[4] = { VERSION5, OK, RESERVED, IP };
352     writen(fd, (void *)response, ARRAY_SIZE(response));
353     writen(fd, (void *)ip, IPSIZE);
354     writen(fd, (void *)&port, sizeof(port));
355 }
356
357 //读取域名
358 char *socks5_domain_read(int fd, unsigned char *size)
359 {
360     unsigned char s;
```

```c
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);
```

```
439             if (nread <= 0)
440                 break;
441             send(fd1, (const void *)buffer_r, nread, 0);
442         }
443
444         if (FD_ISSET(fd1, &rd_set)) {
445             nread = recv(fd1, buffer_r, BUFSIZE, 0);
446             if (nread <= 0)
447                 break;
448             send(fd0, (const void *)buffer_r, nread, 0);
449         }
450     }
451 }
452
453 void *app_thread_process(void *fd)
454 {
455     int net_fd = *(int *)fd;
456     int version = 0;
457     int inet_fd = -1;
458     char methods = socks_invitation(net_fd, &version);
459
460     switch (version) {
461     case VERSION5: {
462             socks5_auth(net_fd, methods);
463             int command = socks5_command(net_fd);
464
465             if (command == IP) {
466                 char *ip = socks_ip_read(net_fd);
467                 unsigned short int p = socks_read_port(net_fd);
468
469                 inet_fd = app_connect(IP, (void *)ip, ntohs(p));
470                 if (inet_fd == -1) {
471                     app_thread_exit(1, net_fd);
472                 }
473                 socks5_ip_send_response(net_fd, ip, p);
474                 free(ip);
475                 break;
476             } else if (command == DOMAIN) {
477                 unsigned char size;
478                 char *address = socks5_domain_read(net_fd, &size);
479                 unsigned short int p = socks_read_port(net_fd);
480
481                 inet_fd = app_connect(DOMAIN, (void *)address, ntohs(
482                 if (inet_fd == -1) {
483                     app_thread_exit(1, net_fd);
484                 }
485                 socks5_domain_send_response(net_fd, address, size, p);
486                 free(address);
487                 break;
488             } else {
489                 app_thread_exit(1, net_fd);
490             }
491         }
492         case VERSION4: {
493             if (methods == 1) {
494                 char ident[255];
495                 unsigned short int p = socks_read_port(net_fd);
496                 char *ip = socks_ip_read(net_fd);
497                 socks4_read_nstring(net_fd, ident, sizeof(ident));
498
```

```
                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) {
        if (pthread_create
            (&worker, NULL, &app_thread_process,
             (void *)&net_fd) == 0) {
```

```
582                 (void *)&net_fd) == 0) {
583                 pthread_detach(worker);
584             } else {
585                 log_message("pthread_create()");
586             }
587         }
588     }
589
590     void daemonize()
591     {
592         pid_t pid;
593         int x;
594
595         pid = fork();
596
597         if (pid < 0) {
598             exit(EXIT_FAILURE);
599         }
600
601         if (pid > 0) {
602             exit(EXIT_SUCCESS);
603         }
604
605         if (setsid() < 0) {
606             exit(EXIT_FAILURE);
607         }
608
609         signal(SIGCHLD, SIG_IGN);
610         signal(SIGHUP, SIG_IGN);
611
612         pid = fork();
613
614         if (pid < 0) {
615             exit(EXIT_FAILURE);
616         }
617
618         if (pid > 0) {
619             exit(EXIT_SUCCESS);
620         }
621
622         umask(0);
623         chdir("/");
624
625         for (x = sysconf(_SC_OPEN_MAX); x >= 0; x--) {
626             close(x);
627         }
628     }
629
630     void usage(char *app)
631     {
632         printf
633             ("USAGE: %s [-h][-n PORT][-a AUTHTYPE][-u USERNAME]
634              app);
635         printf("AUTHTYPE: 0 for NOAUTH, 2 for USERPASS\n");
636         printf
637             ("By default: port is 1080, authtype is no auth, lo
638         exit(1);
639     }
640
641     int main(int argc, char *argv[])
642     {
643         int ret;
644         log_file = stdout;
645         auth_type = NOAUTH;
646         arg_username = "user";
647         arg_password = "pass";
648         pthread_mutex_init(&lock, NULL);
649
650         signal(SIGPIPE, SIG_IGN);
651
652         while ((ret = getopt(argc, argv, "n:u:p:l:a:hd")) != -1
653             switch (ret) {
654             case 'd':{
655                     daemon_mode = 1;
656                     daemonize();
657                     break;
658                 }
659             case 'n':{
660                     port = atoi(optarg) & 0xffff;
658                 }
659             case 'n':{
660                     port = atoi(optarg) & 0xffff;
```

```
660             port = atoi(optarg) & 0xffff;
661             break;
662         }
663     case 'u':{
664             arg_username = strdup(optarg);
665             break;
666         }
667     case 'p':{
668             arg_password = strdup(optarg);
669             break;
670         }
671     case 'l':{
672             freopen(optarg, "wa", log_file);
673             break;
674         }
675     case 'a':{
676             auth_type = atoi(optarg);
677             break;
678         }
679     case 'h':
680     default:
681         usage(argv[0]);
682     }
683 }
684 log_message("Starting with authtype %X", auth_type);
685 if (auth_type != NOAUTH) {
686     log_message("Username is %s, password is %s", arg_username,
687             arg_password);
688 }
689 app_loop();
690 return 0;
691 }
```

3 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