

****

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1班**

**姓　　名 雷鸿宇**

**学　　号 22920192204173**

**实验时间 2021年6月1日**

**2021 年 6 月 1 日**

填写说明

1. 本文件为Word模板文件，建议使用Microsoft Word 2019打开，在可填写的区域中如实填写；
2. 填表时，勿破坏排版，勿修改字体字号，打印成PDF文件提交；
3. 文件总大小尽量控制在1MB以下，勿超过5MB；
4. 应将材料清单上传在代码托管平台上；
5. 在学期最后一节课前按要求打包发送至cni21@qq.com。

# 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。

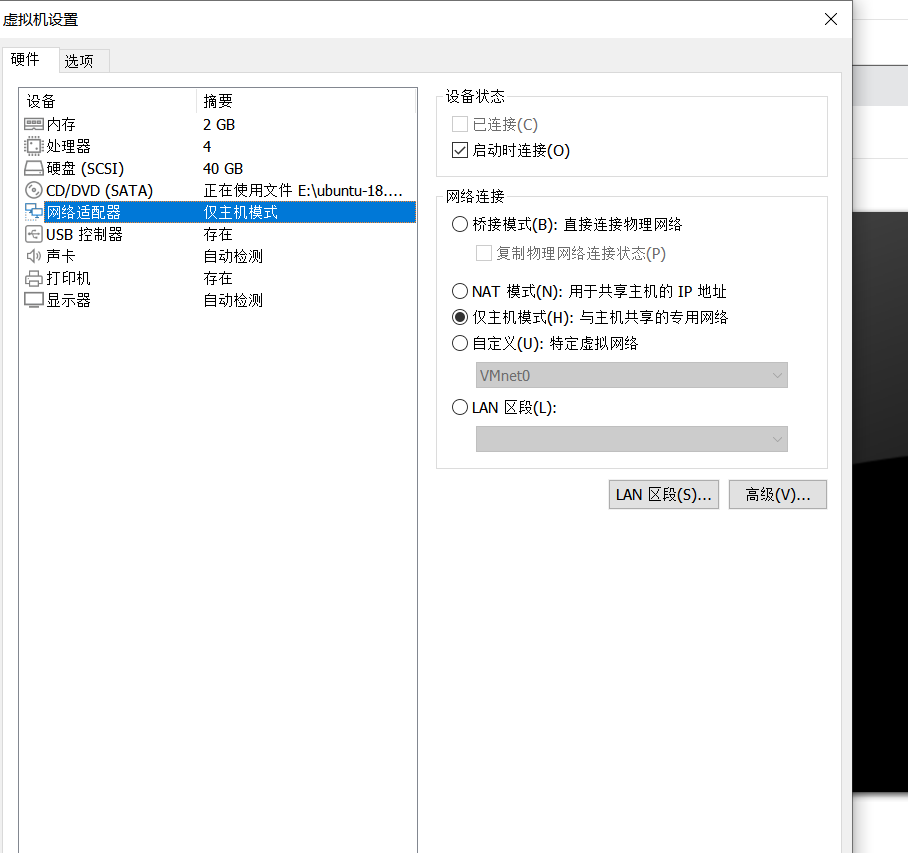
# 实验环境

# Vmware linux ubuntu

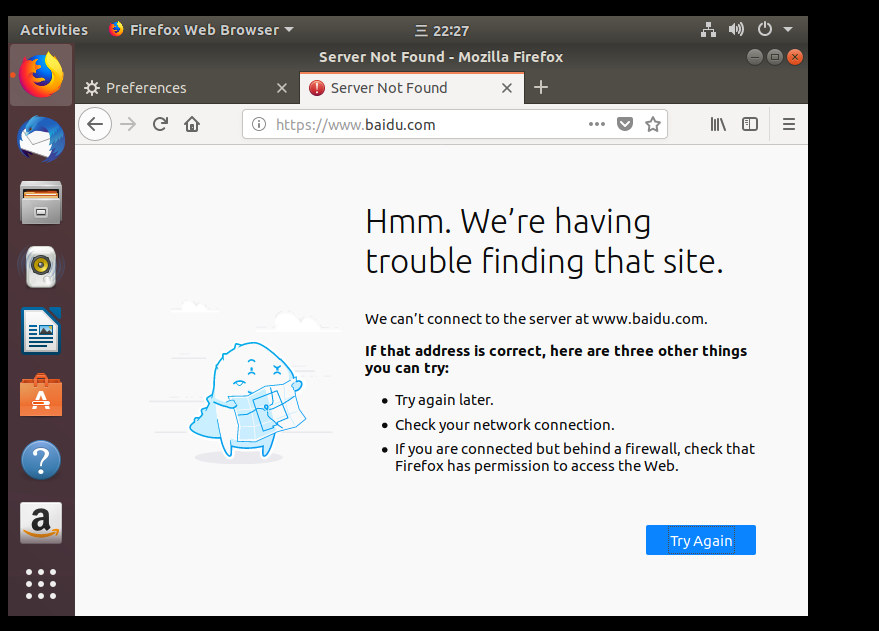
# c语言

# 实验结果

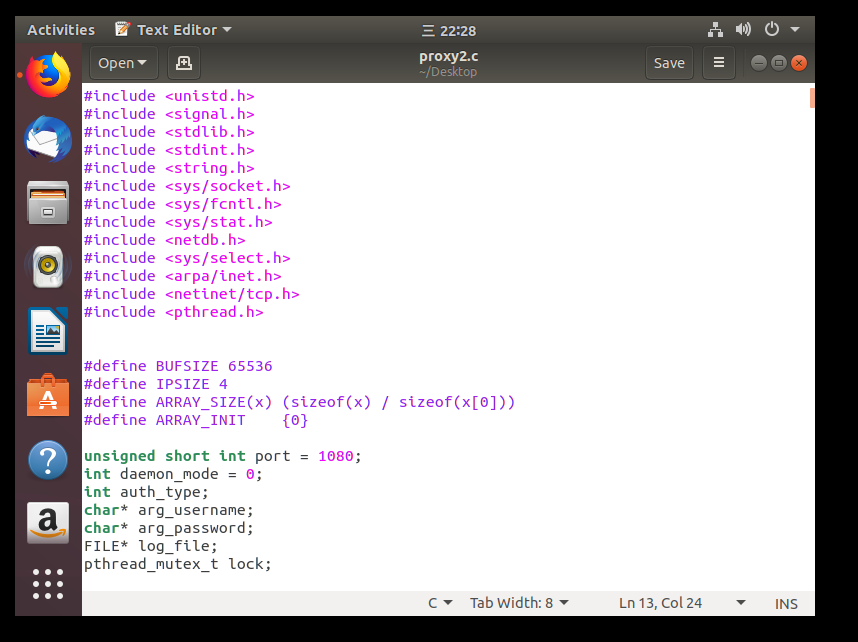
# Vmware虚拟机网络连接调整为host-only



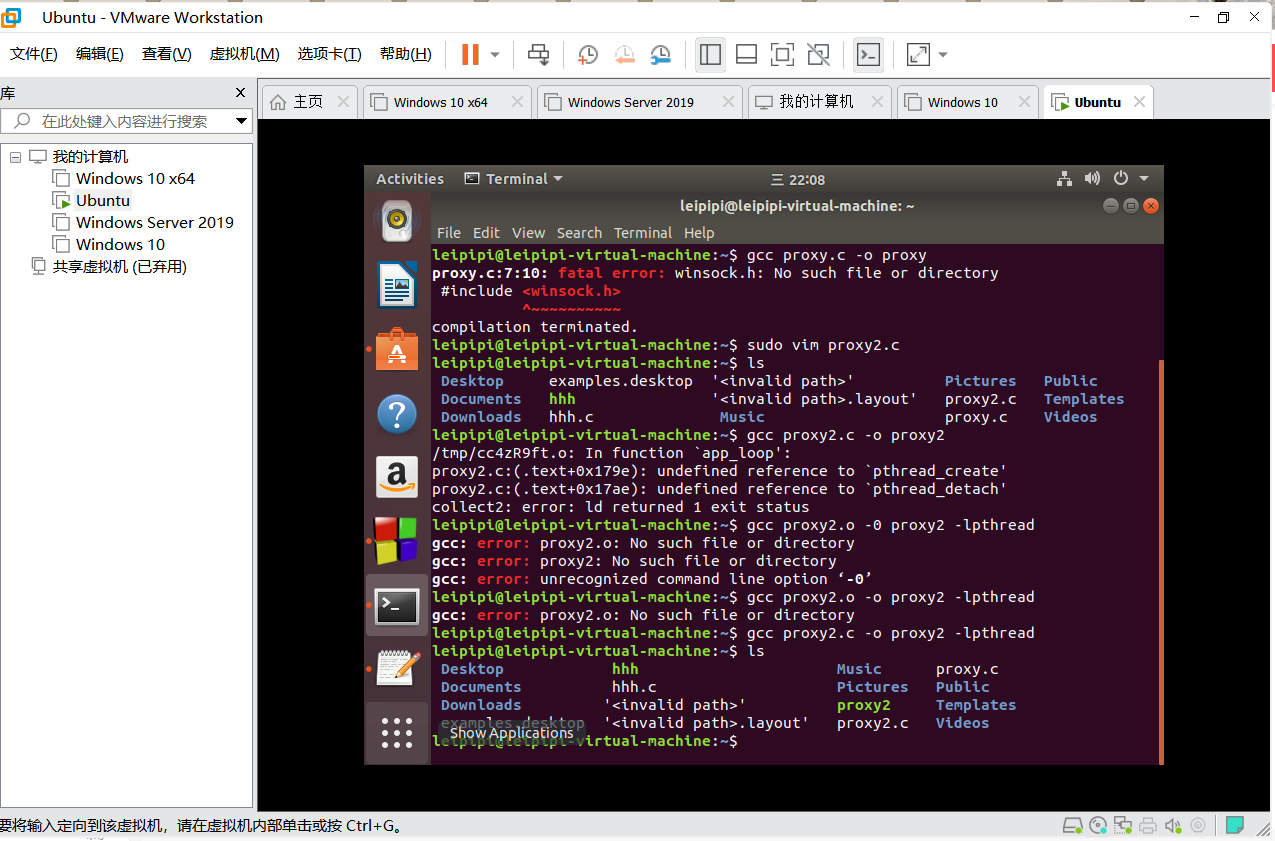
可以看到，目前无法访问



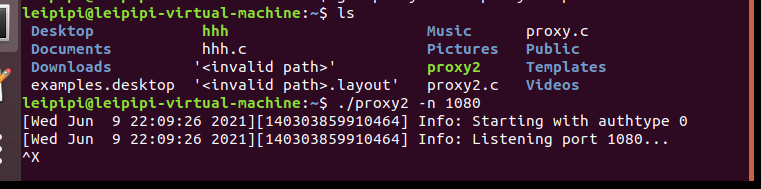
使用proxy示例代码(已对源代码进行阅读并进行相应注释 )



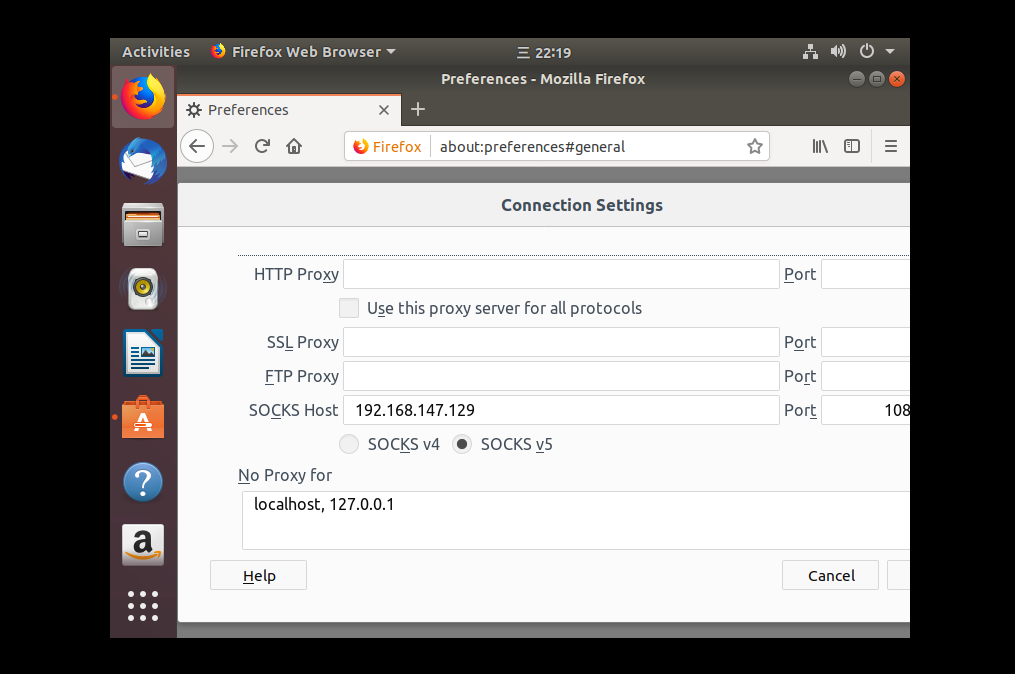
使用vim编译代码



运行程序，使其监听1080端口



调整浏览器设置



部分代码（含注释）

unsigned short int port = 1080;//端口号

int daemon\_mode = 0;

int auth\_type;

char\* arg\_username;//用户名

char\* arg\_password;//密码

FILE\* log\_file;//日志文件，记录运行情况

pthread\_mutex\_t lock;//pthread库的一个线程锁

//状态参数

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);

va\_end(args);

time\_t now;

time(&now);

char\* date = ctime(&now);

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

pthread\_t self = pthread\_self();//获取自身线程

if (errno != 0) {

pthread\_mutex\_lock(&lock);//操作前将互斥锁上锁

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);//错误

}

fflush(log\_file);

}

int readn(int fd, void\* buf, int n)//读取内容

{

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;

}

int writen(int fd, void\* buf, int n)//写入内容

{

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;

}

void app\_thread\_exit(int ret, int fd)//关闭线程连接和socket

{

close(fd);

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

}

int app\_connect(int type, void\* buf, unsigned short int portnum)//建立线程连接

{

int fd;

struct sockaddr\_in remote;

char address[16];

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

//ip模式下

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));//初始化然后获取ip地址和端口

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;

}

//域模式下

else if (type == DOMAIN) {

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);//服务到端口的转换，存入res

if (ret == EAI\_NODATA) {//(No address associated with hostname

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));

//若版本与5和4都不兼容

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]);//初始化版本成功

\*version = init[0];

return init[1];

}

# 实验代码

本次实验的代码已上传于以下代码仓库：<https://github.com/leipipi>

# 实验总结

本实验尝试设计一个代理服务器软件，由于难度过大，时间不足，于是对示例代码进行了阅读注释，并尝试在linux上运行代码实现其功能。明白了如何基于socket进行代理服务器软件编程，实现了在linux上使浏览器通过该软件能够成功进行网页访问。同时还涉及了一些线程知识，加深了对线程应用的理解。