哈爾濱工業大學

网络安全实验报告

题　　目 基于 socket 的扫描器设计

专　　业 计算机科学与技术

学　　号 7203610316

学　　生 符兴

指 导 教 师 王彦

一、实验目的

熟悉 socket 编程，可以利用 socket 编程编写基于 linux 平台的 C/S 程序和基于 windows 平台的扫描器。

二、实验内容

1.熟悉 Linux 编程环境

2.在 Windows 机器上安装 Linux 虚拟机

3.在 Linux 环境下编写 C/S 程序，熟悉 socket 编程。要求客户端和服务器端能够传送指定文件。该程序在后续实验中仍需使用。客户端与服务器端在不同的机器中。

4.在 Windows 环境下利用 socket 的 connect 函数进行扫描器的设计，要求有界面，界面能够输入扫描的 ip 范围和端口范围，和需使用的线程数，显示结果。

5.实验课的时候，检验结果和现场截图，为撰写实验报告做准备。

三、实验过程

（一）Linux 环境下的 C/S 程序

实验基本信息：

实验环境：Ubuntu 20.04 x64 编程语言：C

1. 需求分析

需要在两台 linux 虚拟机之间传送文件，所以需要给两台 linux 虚拟机都配置一个可以访问的 ip。

程序功能：

(1)客户端：

a.可以向服务端发送一个本目录下指定的文件，文件名由用户输入；

b.可以从服务端下载一个服务端目录下的文件，先从服务端获得文件名列表，再由用户输入需要的文件名。

(2)服务端：

a.可以监听来自客户端的连接请求；

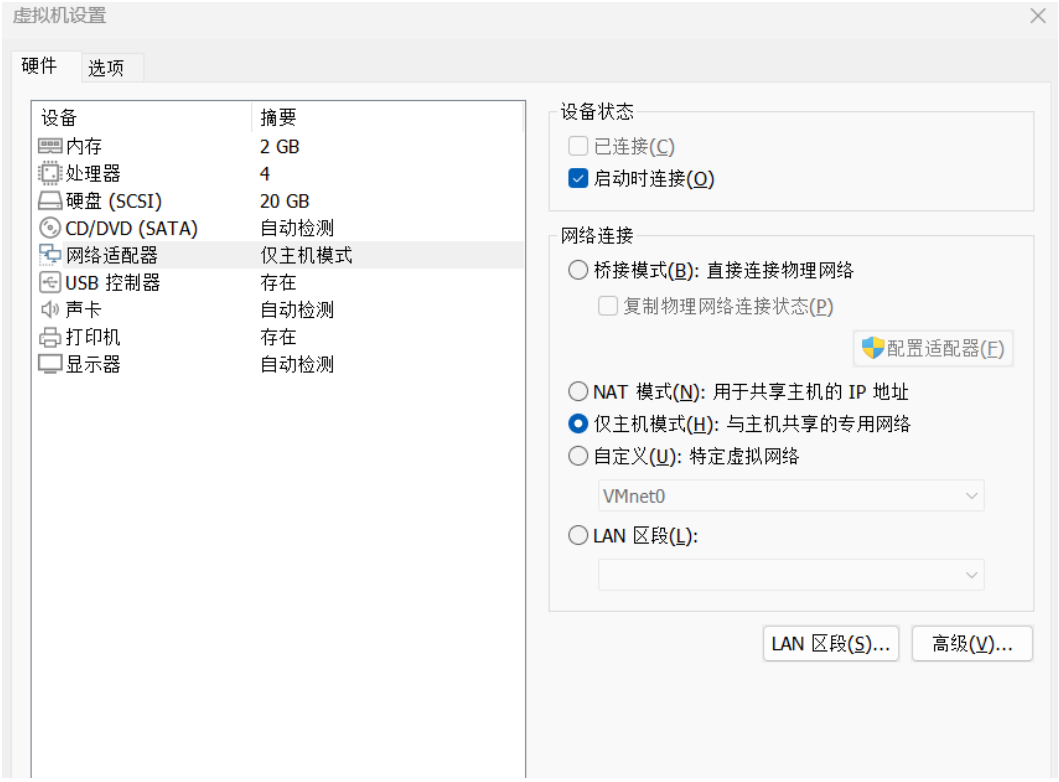
b.可以接收客户端传送的文件；

c.可以向客户端传送一个指定的文件，文件由客户端给出。

(3)传送文件要求：任何二进制文件。

2. 环境配置

配置虚拟机的网卡即可：



3. 客户端编写

```
int main(int argc, char **argv)
{
    if (argc != 3)
    {
        printErrorInfo("请正确输入服务器参数，包括IP地址和端口号");
        exit(-1);
    }
    char *addr = argv[1];
    int port = atoi(argv[2]);

    // 创建套接字
    int socketFd = socket(AF_INET, SOCK_STREAM, 0);
    if (socketFd == -1)
    {
        printErrorInfo("创建套接字失败，请重试");
        exit(-1);
    }

    // 连接服务器
    struct sockaddr_in serverInfo;
    memset(&serverInfo, 0, sizeof(serverInfo));
    serverInfo.sin_family = AF_INET;
    serverInfo.sin_addr.s_addr = inet_addr(addr);
    serverInfo.sin_port = htons(port);
    int status = connect(socketFd, (struct sockaddr *)&serverInfo, sizeof(serverInfo));
    if (status == -1)
    {
        printErrorInfo("连接服务器失败，请检查IP地址或端口号是否正确");
        exit(-1);
    }
    printWelcomInfo();

    // 选择功能
    int funcID;

    printf("请键入你需要的功能: ");
    scanf("%d", &funcID);
```

其中，客户端所要连接的服务器信息通过超参给出，即./client –addr –port;在客户端连接上服务器后，客户端需要选择所要进行的功能。

如果选择上传文件的功能，则程序会首先发送功能序号，当收到服务器发回的 Recv 信息后，服务器将构建一个文件上传帧，其中包括文件名和文件的总字节数，并将该文件上传帧发送给服务器进行确认。当收到服务器的 Recv 信息后，客户端才开始发送文件。

如果选择下载文件的功能，程序依旧是先发送功能序号，然后接收服务器上所存储的文件信息列表。然后客户需要选中并发送需要接收的文件名，等待服务器回传文件信息帧，其中包括文件名和文件的总字节数，然后发送 Recv 信息给服务器，告知服务器已经准备好接收文件。

```
// 选择需要发送的文件
    char fileName[BUFFER_SIZE];
    memset(&fileName, 0, BUFFER_SIZE);
    printf("请输入需要发送的文件名:");
    scanf("%s", fileName);

    // 读取文件并发送
    FILE *file = fopen(fileName, "rb");
    if (file == NULL)
    {
        printErrorInfo("读取文件失败");
        exit(-1);
    }

    // 构建响应帧
    union responseFrame2Char frame;
    memset(&frame.data, 0, BUFFER_SIZE);
    memcpy(frame.dataFrame.fileName, fileName, strlen(fileName));
    frame.dataFrame.size = getSize(file);
    printf("待发送文件的总字节数为:%d\n", frame.dataFrame.size);

    memset(&sendBuffer, 0, BUFFER_SIZE);
    memset(&recvBuffer, 0, BUFFER_SIZE);
    memcpy(sendBuffer, frame.data, BUFFER_SIZE);
    sendLen = write(socketFd, sendBuffer, BUFFER_SIZE);
    recvLen = read(socketFd, recvBuffer, BUFFER_SIZE);
    if (strcmp(recvBuffer, "Recv.") != 0)
    {
        printErrorInfo("传送文件时候出现错误");
        exit(-1);
    }

    int totalSendLen = 0;
    while (!feof(file))
    {
        memset(&sendBuffer, 0, BUFFER_SIZE);
        readLen = fread(sendBuffer, sizeof(char), BUFFER_SIZE, file);
        sendLen = write(socketFd, sendBuffer, readLen);
        totalSendLen += sendLen;
        printf("已发送字节数: %d\n", totalSendLen);
    }
```

图 客户端发送文件给服务器

```
// 发送功能序号
    memset(&sendBuffer, 0, BUFFER_SIZE);
    memset(&recvBuffer, 0, BUFFER_SIZE);
    strcpy(sendBuffer, "2");
    sendLen = write(socketFd, sendBuffer, BUFFER_SIZE);
    recvLen = read(socketFd, recvBuffer, BUFFER_SIZE);
    printf("这是服务器上的文件列表\n");
    printf("%s\n", recvBuffer);

    memset(&recvFileName, 0, BUFFER_SIZE);
    printf("请选择你要下载的文件名: ");
    scanf("%s", recvFileName);

    // 发送接收的文件名
    memset(&sendBuffer, 0, BUFFER_SIZE);
    strncpy(sendBuffer, recvFileName, strlen(recvFileName));
    sendLen = write(socketFd, sendBuffer, BUFFER_SIZE);

    // 接收文件名称
    memset(&recvBuffer, 0, BUFFER_SIZE);
    recvLen = read(socketFd, recvBuffer, BUFFER_SIZE);
    union responseFrame2Char frame;
    memset(&frame.data, 0, BUFFER_SIZE);
    memcpy(frame.data, recvBuffer, BUFFER_SIZE);
    strcpy(recvFileName, frame.dataFrame.fileName);
    printf("待接收文件的总字节数为:%d\n", frame.dataFrame.size);

    memset(&sendBuffer, 0, BUFFER_SIZE);
    strcpy(sendBuffer, "Recv.");
    sendLen = write(socketFd, sendBuffer, BUFFER_SIZE);
    usleep(1e6);
```

```
    // 接收文件
    char *tmpPrefix = "tmp_";
    char *name = (char *)malloc(strlen(tmpPrefix) + strlen(recvFileName))
    strcat(name, tmpPrefix);
    strcat(name, recvFileName);
    printf("%s\n", name);
    FILE *file = fopen(name, "wb");
    int totalRecvLen = 0;
    while (1)
    {
        memset(&recvBuffer, 0, BUFFER_SIZE);
        recvLen = read(socketFd, recvBuffer, BUFFER_SIZE);
        fwrite(recvBuffer, sizeof(char), recvLen, file);
        totalRecvLen += recvLen;
        if (totalRecvLen >= frame.dataFrame.size)
        {
            break;
        }
    }
```

图 客户端从服务器下载文件

4. 服务端编写

服务端在创建后将监听连接，针对每个连接都会单开一个线程去进行处理。

如果客户端的功能是上传文件，服务器会先等待接收文件上传帧，确定所需接收文件的总字节数；然后发送 Recv 告知客户端已经准备好接收文件了。

如果客户端需求功能是从服务器下载文件，服务器首先会将其存储的文件列表发送给客户端，然后等待接收客户端所需要的文件；当收到客户端的文件需求后，构建文件下载帧，将文件名和文件总字节数发送给客户端，并等待客户端的进一步确认；当收到确认后，服务端正式开始发送文件。

```
// 监听客户端
    while (1)
    {
        struct sockaddr_in clientAddr;
        int len = sizeof(clientAddr);
        int clientFd = accept(socketFd, (struct sockaddr *)&clientAddr, &len);
        if (clientFd == -1)
        {
            printErrorInfo("与客户端连接失败");
            continue;
        }
        printf("收到一个新的客户端连接\n");
        struct client tmp_client;
        tmp_client.clientFd = clientFd;
        tmp_client.clientAddr = &clientAddr;
        pthread_t id;
        // 创建子线程
        int ret = pthread_create(&id, NULL, (void *)process, (void *)&tmp_client);
        if (ret != 0)
        {
            printErrorInfo("与客户端连接失败");
        }
    }

```

图 服务器监听连接

（二）Windows 环境下的扫描器程序

实验基本信息：

实验环境：Windows10 x64

QtCreator 4.8.1 编程语言：C++

1. 需求分析

实验指导中要求编写界面，可以使用 java，但是 java 编写界面过于麻烦，所以我选择了基于 C++的 QtCreator 来编写程序，QtCreator 的界面编写非常方便（拖拖拖），且它独有的信号与槽机制能使很多操作变得方便。

另外，在程序的设计各方面都追求人性化，用户误操作时会给出准确的提示息。

程序功能：

(1) 用户可以输入需要扫描的 ip 范围、端口范围和想使用的线程数，其中 ip 范围跟平时在电脑上操作一样，输入三个数字后自动跳转到下一个输入框，输入框中只能输入合法的字符；

(2) 如果用户在输入未完成的时候就按下了开始扫描按钮，提示输入未完成，如果用户输入的范围错误，提示范围错误；

(3) 当所有输入都正确无误后，按下开始扫描，程序开始扫描用户指定的 ip 和端口；

(4) 关于扫描的线程分配：

方案一：由于本人技术有限，采取先把 ip 和端口号一对一保存，根据 step=总端口数/线程数 给每个线程分配 step 个端口（最后一个线程扫描剩下所有端口）。但是这种方法有一个弊端，例如 1000 个端口，300 个线程，前 299 个线程每个线程只用扫描 3 个端口，最后一个线程却需要扫描剩下的 103 个端口，这显然不符合多线程的初衷，于是我改进了分配端口的方法。

方案二：前面与方案一相同，但每给一个线程分配好端口数后，就计算一次剩下的端口/剩下的线程，如果这个值大于 step，就表示之后的每一个线程需要多分配几个端口（准确地说是 1 个），则将其赋给 step，测试程序后发现运行速度明显提高了，不存在一个线程扫描超多端口的现象。

(5) 关于扫描输出：本着用户友好原则，在扫描过程中打印所有的扫描结果，但因为多线程的原因，扫描出的顺序是乱的，所以在扫描结束后单独打印出开启的端口号，并且打印此次扫描花费的时间、扫描的总端口数以及开启的端口数。

(6) 用户可以在扫描正在进行时按结束扫描的按钮来中断扫描，点击按钮后会跳出对话框确认以防止用户误点，当程序收到结束扫描的信号时会中断所有线程，这一过程是安全的。

2. 界面编写

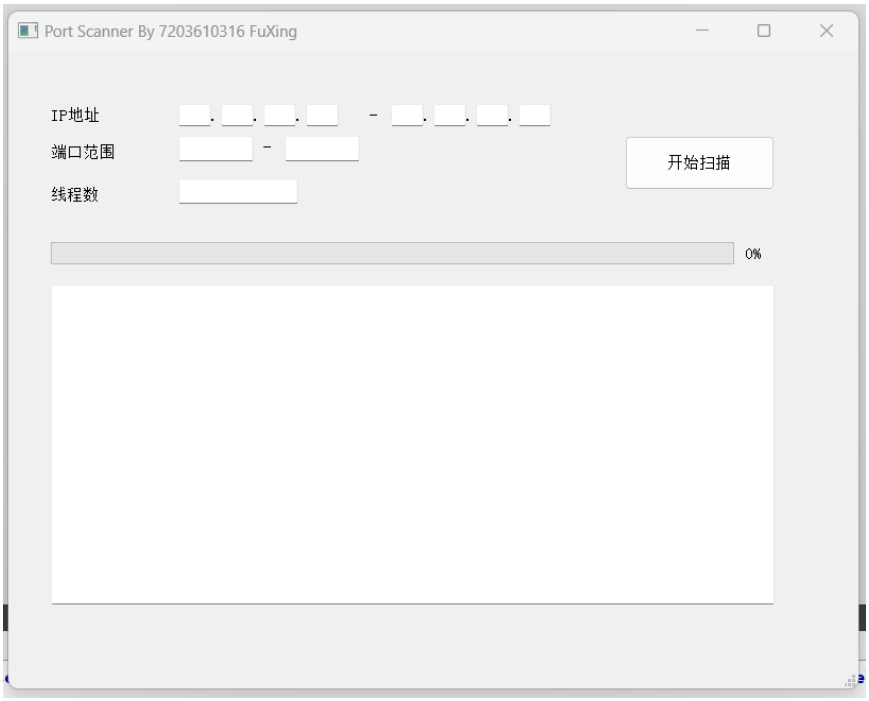


图 端口扫描器的图形化界面

整个界面大致包括三个部分：信息输入区、功能区和信息输出区。

其中 IP 地址的输入组件由于需要监听分隔符进行移动，所以 IP 地址的输入组件在继承 QT 的 LineEdit 组件基础上重新封装了监听逻辑。其中 m_next 指向下一个输入组件，当用户在输入 IP 地址时，程序会自动地将当前 focused Object 指向 m_next；

```
class LineEdit:public QLineEdit{
    Q_OBJECT;
public:
    explicit LineEdit(QWidget* parent = 0);
    void setValidCheck();
    void keyPressEvent(QKeyEvent * event) override;
    void focusInEvent(QFocusEvent *event) override;
    void setNext(LineEdit* next){
        this->m_next = next;
    }

private:
    QLineEdit* m_next;
};
```

图 IP 地址输入组件

3. 控件逻辑编写

目前各个组件之间的控制逻辑有：

1. 点击“扫描”按钮进行扫描。

该按钮通过 QT 的 connect()将 clicked()行为和函数 startJob()进行绑定。当用户点击按钮时，程序会判断当前是否正在进行扫描任务，如果未存在扫描任务则调用 startIPScan()函数；如果当前正在进行扫描，程序会给出“是否终止扫描任务”的警告，如果用户选择是，则终止扫描并输出终止信息；如果用户选择否，则程序会继续当前未完成的扫描任务。

```
void MainWindow::startJob(){
    if(m_isRunner == 0){
        startIPScan();
    }else{
        QMessageBox::StandardButton qBox = QMessageBox::question(this,"提示", "确定要终止扫描程序吗?", QMess
        if(qBox == QMessageBox::Yes){
            emit sendStopSignal();
            m_ui->progressBar->setValue(0);
            m_ui->startBtn->setText("扫描");
            m_isRunner = 0;
            m_ui->textBrowser->append(QString("============================================="));
            m_ui->textBrowser->append(QString("STOP"));
            m_ui->textBrowser->append(QString("============================================="));
        }
    }
}

void MainWindow::bindButtonFuc(){
    m_ui->progressBar->setMinimum(0);
    m_ui->progressBar->setMaximum(100);
    m_ui->progressBar->setValue(0);
    connect(m_ui->startBtn,SIGNAL(clicked(bool)),this,SLOT(startJob()));
}
```

图 点击“扫描”按钮进行扫描。

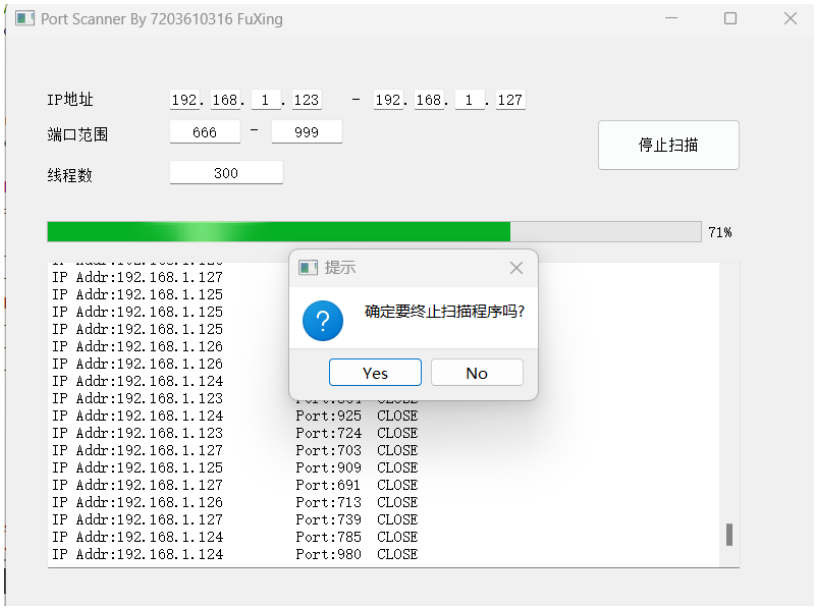


图 是否中断扫描

2. 开始进行扫描时的参数检查

在开始扫描前，程序会检查用户的参数是否非空以及是否满足限制。如果参数不满足限制则会给出相应的警告，告知用户当前输入的参数存在错误。

```
void MainWindow::checkIPNum(QString data){
    int num = data.toInt();
    if(num>255){
        QMessageBox::critical(this, tr("错误"),  tr("最大值为255"));
    }
}

void MainWindow::checkPortNum(QString data){
    int num = data.toInt();
    if(data == ""){
        return ;
    }
    if(num>=65535){
        QMessageBox::critical(this, tr("错误"),  tr("最大值为65535"));
    }else if(num<0){
        QMessageBox::critical(this, tr("错误"),  tr("最小值为10"));
    }
}

void MainWindow::checkThreadNum(QString data){
    int num = data.toInt();
    if(data == ""){
        return ;
    }
    if(num>=1000){
        QMessageBox::critical(this, tr("错误"),  tr("最大值为1000"));
    }else if(num<0){
        QMessageBox::critical(this, tr("错误"),  tr("最小值为1"));
    }
}
```

图 参数检查

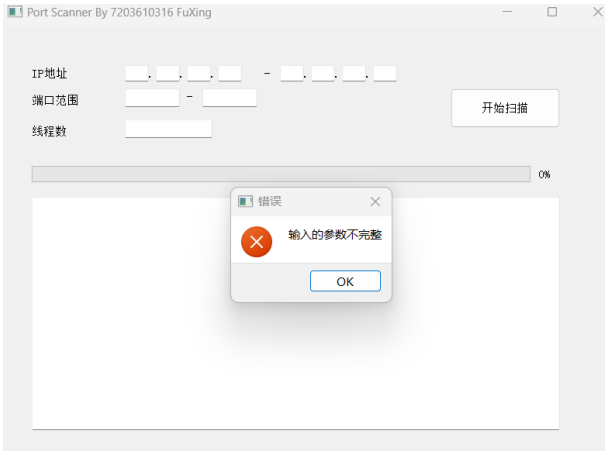


图 参数错误程序给出警告

3. 扫描过程中给出进度信息

在执行扫描任务中，程序会通过进度条的方式告知用户当前任务完成的进度。

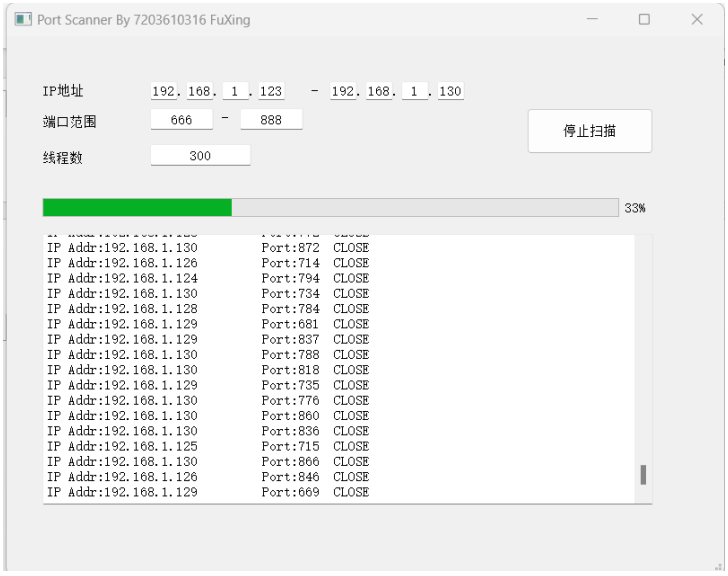


图 进度条

4. 扫描完成信息

在扫描完成后，程序会给出端口开放的信息和扫描任务的总耗时信息。

4. 具体功能编写

(1) 扫描主方法 void MainWindow::applyJob(long ipv4S, long ipv4E, int portS, int portE, int threadNum, QTextBrowser* textUi)

```
void MainWindow::applyJob(long ipv4S, long ipv4E, int portS, int portE, int threadNum, QTextBrowser* textUi)
{
    qDebug()<<"IPScanner Apply";
    int lastJobCnt = (ipv4E - ipv4S +1)*(portE-portS+1);
    totalJob = lastJobCnt;
    int lastThreadCnt = threadNum;
    int jobPerThread = lastJobCnt/lastThreadCnt;
    std::vector<std::pair<long,int>> jobList;

    int threadID = 0;
    for(long i = ipv4S; i<=ipv4E; i++){
        for(int j = portS; j<=portE; j++){
            jobList.push_back(std::pair<long,int>(i,j));
            lastJobCnt -= 1;
            if(jobList.size()>=jobPerThread){
                m_scanfThread[threadID] = new ScanfThreadWork(jobList);
                connect(m_scanfThread[threadID],SIGNAL(sendResult(QString,QString,int,int)),this,SLOT(printResult(QString,Q
                connect(this, &MainWindow::sendStopSignal,m_scanfThread[threadID],&ScanfThreadWork::recvStopSignal);
                m_scanfThread[threadID]->start();
                threadID += 1;
                lastThreadCnt -= 1;
                std::vector<std::pair<long,int>>().swap(jobList);
                if(lastJobCnt > 0){
                    jobPerThread = lastJobCnt/lastThreadCnt;
                }
            }
        }
    }
    if(jobList.size()!=0){
        m_scanfThread[threadID] = new ScanfThreadWork(jobList);
        connect(m_scanfThread[threadID],SIGNAL(sendResult(QString,QString,int,int)),this,SLOT(printResult(QString,QString,i
        connect(this, &MainWindow::sendStopSignal,m_scanfThread[threadID],&ScanfThreadWork::recvStopSignal);
        m_scanfThread[threadID]->start();
        std::vector<std::pair<long,int>>().swap(jobList);
    }
    qDebug()<<"Finished Job Sent To Thread.";
}
```

图 扫描主方法

在进行扫描任务时，根据前面讲述的线程分配方法，给每个线程分配任务，并设置好子线程和主线程之间的沟通方式。当子线程完成扫描时会将扫描发送给主线程，由主线程绘制到 ui 中。当用户要终止扫描时，主线程会向子线程发送终止信号。

(2) 扫描线程方法 void ScanfThreadWork::apply()

```
void ScanfThreadWork::apply()
{
    for(auto i = m_job.begin(); i!=m_job.end() && m_isStop;i++){
        int jobIp = i->first;
        int jobPort = i->second;
        int ip1 = (jobIp & 0xff000000) >> 24;
        int ip2 = (jobIp & 0x00ff0000) >> 16;
        int ip3 = (jobIp & 0x0000ff00) >> 8;
        int ip4 = (jobIp & 0x000000ff);
        QString ipAddr = QString("%1.%2.%3.%4")
                .arg(QString::number(ip1))
                .arg(QString::number(ip2))
                .arg(QString::number(ip3))
                .arg(QString::number(ip4));
        io_service ios;
        tcp::socket s(ios);
        tcp::endpoint ep(boost::asio::ip::address_v4::from_string(ipAddr.toStdString()), jobPort);
        if (!aysnc_connect(ios, s, ep, 1000))
        {
            emit sendResult(QString("IP Addr:"+ipAddr+"\tPort:"+QString::number(jobPort)+"\tCLOSE"),ipAddr,jobPort,0
        }else{
            emit sendResult(QString("IP Addr:"+ipAddr+"\tPort:"+QString::number(jobPort)+"\tOPEN"),ipAddr,jobPort,1)
        }
    }
}
```

图 扫描子线程方法

子线程继承了 QT 的 QThread 类，在此基础上封装了扫描的子线程类。子线程在收到任务后，子线程首先将 Int 类型的 IP 地址转为字符串的形式，然后通过 TCP 连接判断端口是否开放；在进行 TCP 连接的过程中，设置一个定时器，如果定时器超时则判断当前端口未开放，并将该扫描结果发送给主线程。同时，子线程在进行的过程中会监听 m_isStop 信号，当主线程发送终止信号后，子线程则会结束当前任务。

(3) 向 TextBrowser 打印扫描结果 void MainWindow::printResult(QString data, QString ipAddr, int jobPort, int status)

```
void MainWindow::printResult(QString data, QString ipAddr, int jobPort, int status){
    if(m_isRunner == 0){
        return;
    }
    m_lock.lock();
    if(status == 1){
        m_opendPort.push_back(std::pair<QString,int>(ipAddr,jobPort));
    }
    m_ui->progressBar->setValue((int)(((float)finishedJob/(float)totalJob)*100));
    m_ui->textBrowser->append(data);
    finishedJob += 1;
    m_lock.unlock();
    if(finishedJob == totalJob){
        m_ui->startBtn->setText("扫描");
        m_ui->progressBar->setValue(100);
        m_endTime = Clock::now();
        long long interval = std::chrono::duration_cast<std::chrono::seconds>(m_endTime - m_startTime).coun

        m_ui->textBrowser->append(QString("============================================="));
        m_ui->textBrowser->append(QString("扫描完成"));
        m_ui->textBrowser->append(QString("用时:"+QString::number(interval)+"s"));
        m_ui->textBrowser->append(QString("开启的端口:"));
        for(auto i:m_opendPort){    ⚠Missing reference in range-for with non trivial type (std::_Vector_it
            m_ui->textBrowser->append(QString(i.first+" "+QString::number(i.second)));
        }
        m_ui->textBrowser->append(QString("============================================="));
    }
}
```

图 向 TextBrowser 打印扫描结果

在输出信息时需要进行加锁操作，保证多线程之间 ui 绘制操作的正常进行。当程序完成后，程序会单独输出用时信息和打开的端口信息。

四、实验结果

(一) Linux 环境下的 C/S 程序

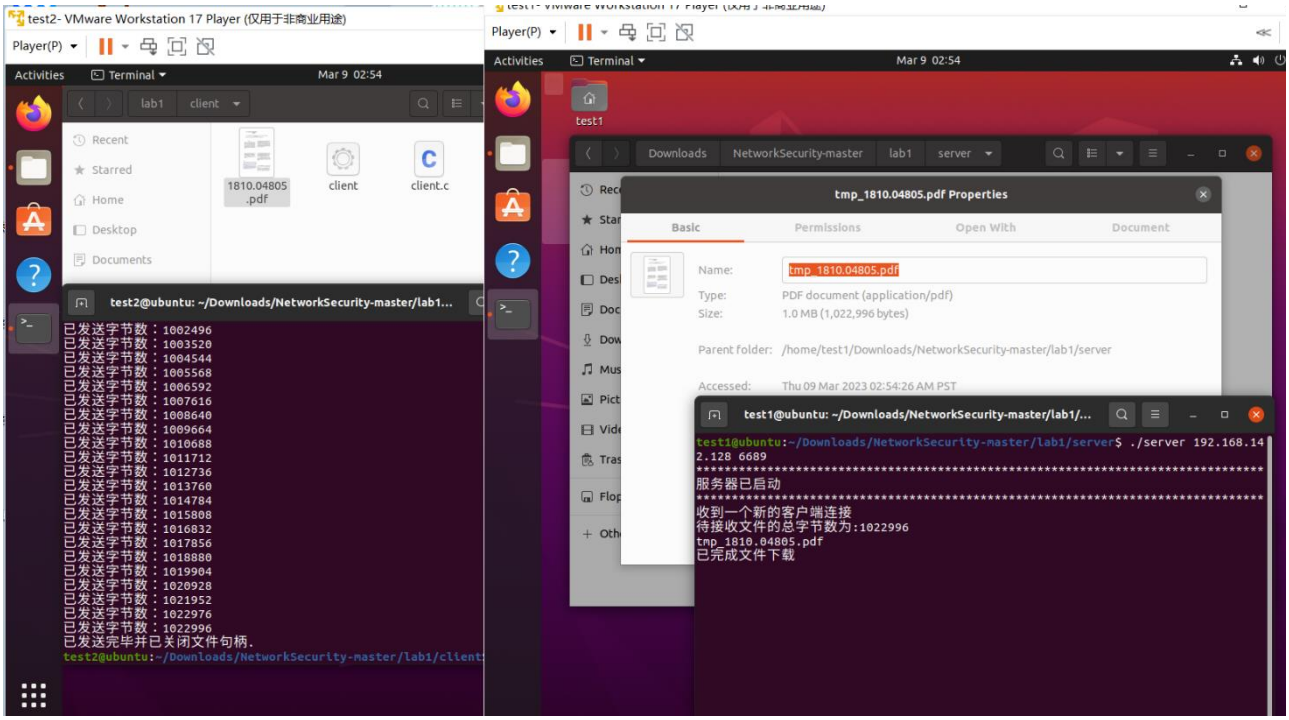


图 客户端上传功能展示

上图中，左边是客户端，右边是服务端；从上图可以看到，文件 1810.04805.pdf 正确地从客户端发往服务器端，其文件的总字节数正确，并且文件可以正确打开。

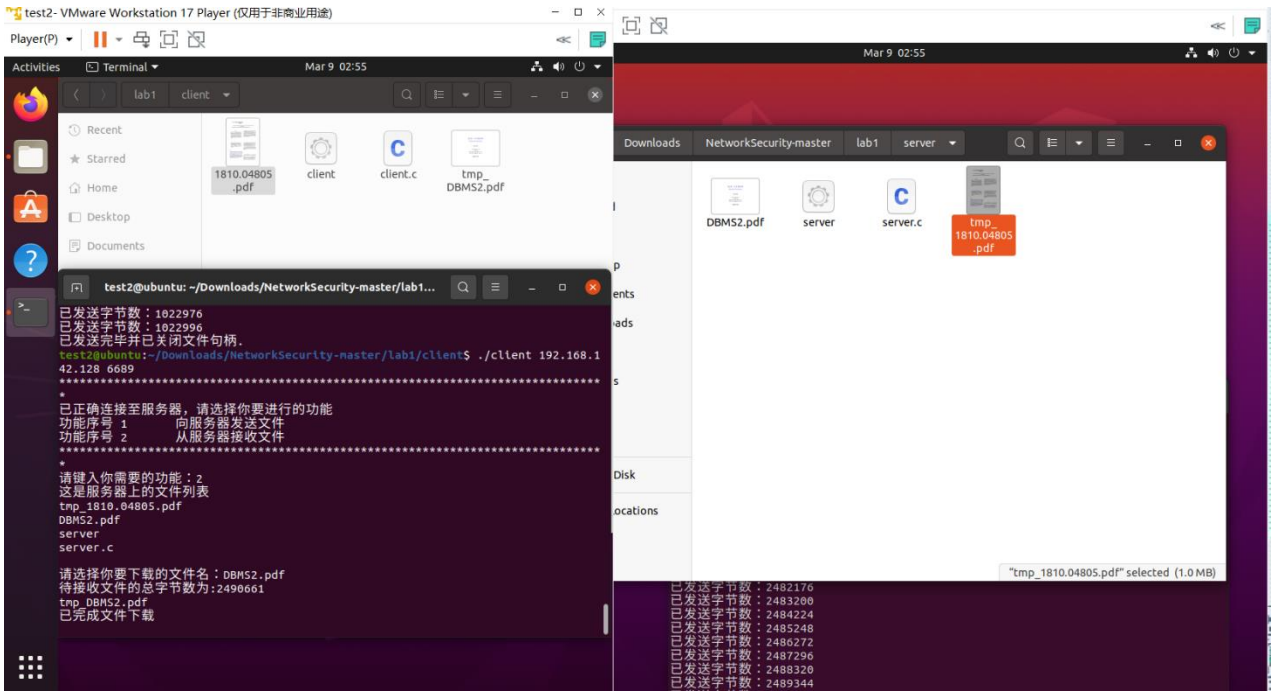


图 客户端从服务器下载文件

上图中，左边是客户端，右边是服务端；从上图可以看到，客户端首先获取到服务器上所存储文件的列表，然后用户键入所要下载的文件。当服务器收到该请求后，将文件发送给客户端，并且客户端正确接收 DBMS2.pdf 文件。

(二) Windows 环境下的扫描器程序

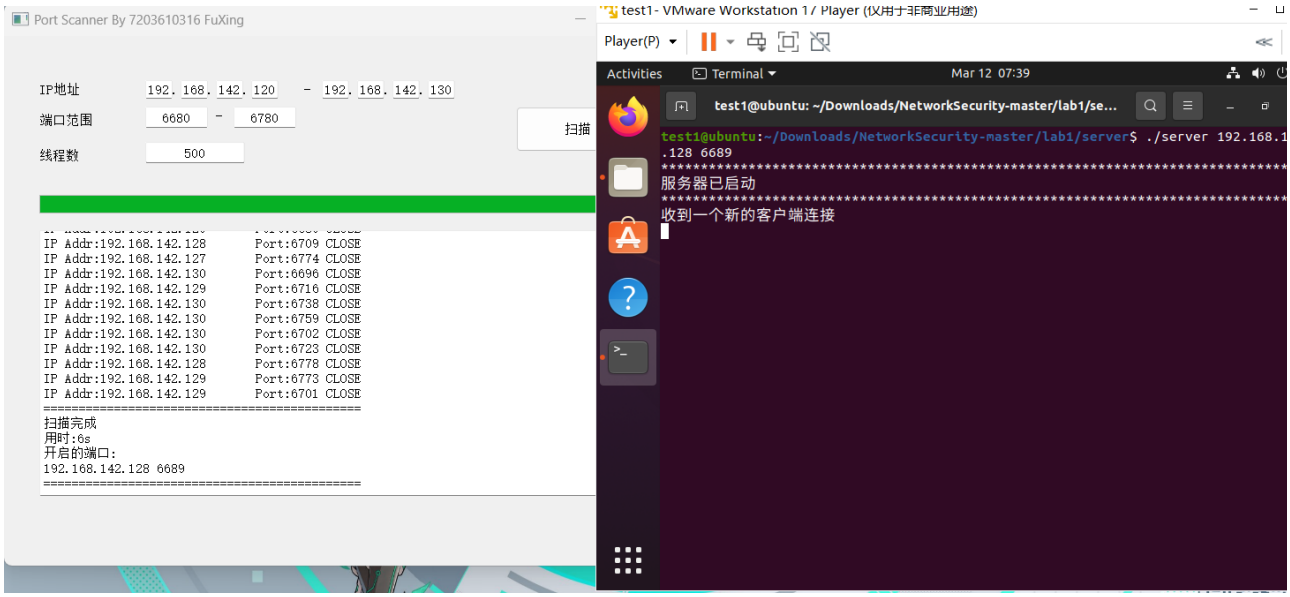


图 扫描实验一文件服务器的端口结果

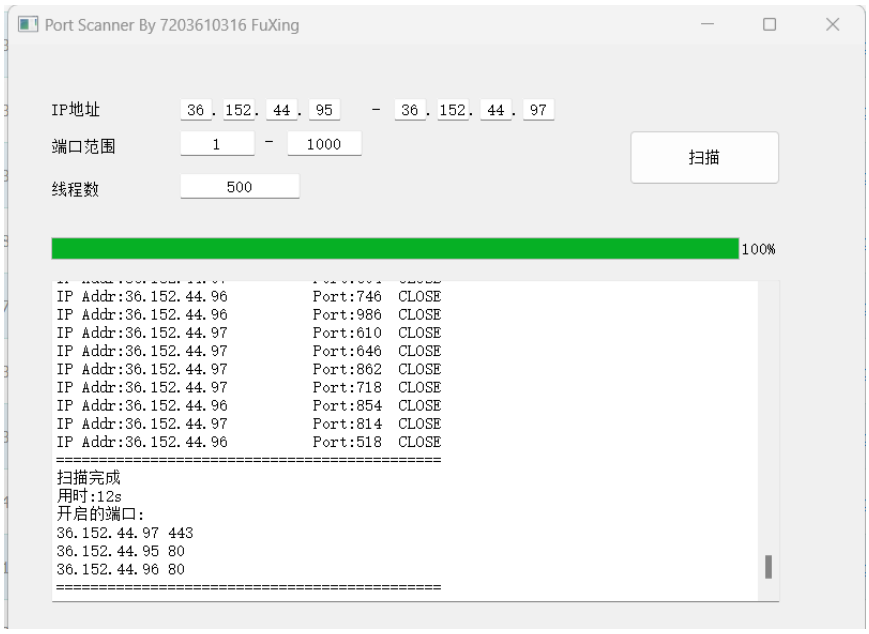


图 扫描百度 IP 端口结果