哈爾濱工業大學

实验报告

实 验（一）

题　　目 基于 socket 的扫描器设计

专　　业 计 算 机 类

学　　号 1160300620

班　　级 1603006

学　　生 张　恒

实 验 地 点 格物 214

实 验 日 期 2019.3.16

计算机科学与技术学院

目 录

第 1 章 实验基本信息

1.1 实验目的

（1）熟悉 Linux 下编程环境;
（2）了解漏洞扫描的基本原理与实现;
（3）熟悉 socket 编程；
（4）熟悉 windows 下的 GUI 编程；

1.2 实验内容

（1） 在 Linux 环境下编写 C/S 程序，要求客户端和服务器端能够传送指定文件，并且客户端与服务器端在不同的机器中；
（2） 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。

1.3 实验环境

本实验分为两个部分，两个部分没有关联，且在不同的环境中，具体如下：

（1） 第一部分的系统环境是 Ubuntu 18.04，编程语言为 C 语言，使用了 g++ 编译器；
（2） 第二部分的系统环境是 Windows 10，编程语言为 Python，使用的是 Python 3.x 的解释器；

第 2 章 实验原理

2.1 C/S 程序的原理

用以传输文件的 C/S 程序，并且客户端和服务器端均能够收发文件，主要涉及到两个比较重要的方面：

一是服务器的设计。既然称作服务器，那么就应该具有服务器的一般属性，即一直在线响应客户端请求。服务器的模型可以简单的分为循环服务器和并发服务器两种，前者在同一时刻只处理一个客户端的请求，后者可以在同一时刻处理多个客户端的请求。考虑到文件传输服务器单次服务的时间比较长，我这里选择并发服务器模型，其原理是利用计算机的多线程（或者是多进程，我这里选择的是利用多线程），将每个客户端请求交给独立的线程处理，可以充分利用服务器端的计算资源。

二是客户端与服务器的交互问题。服务器怎么判断客户端是想传文件还是下载文件？客户端或者服务器端如何判断是已经接收到全部的数据，还是由于网络的延迟，部分数据还没到达？为了解决这些问题，我为每次传输的数据增加了一个六个字节的头部，其结构如图 2-1 所示。

图 2-1 C/S 进行数据交换时的头部结构

其中“请求/响应类型”表示请求或者响应的类型，表明当前数据包是携带何种数据；“是否是最后一个”表示当前数据包是否是最后一个数据分片，在传输文件时，一个文件通常需要划分为多个数据片段来进行传输，这个位用以表明是否已经传输结束。对于普通的命令式交互信息，通常比较短，因此通常置一；“本次数据包长度”表明当前数据包共有多少字节，用以判断是否接受到全部数据。

2.2 端口扫描器的原理

端口扫描器可以利用的原理有很多，包括传统扫描、TCP SYN 扫描、TCP FIN 扫描、分片扫描、FTP 跳转、UDP ICMP 端口不可达等，本实验中选择了传统扫描，该方式的原理是利用系统的 connect()函数连接目标端口，如果函数返回成功就认为端口是开放的。这种方式还可以利用多线程来加速扫描。

为了使扫描器拥有 GUI，我这里选择了 python 自带的 thinter 接口。

第 3 章 程序流程设计

3.1 C/S 程序的流程图

3.1.1 服务器端的流程

本实验中采用了并发服务器的模型，并且选择多线程来实现并发。服务器的流程如图 3-1 所示，服务器监听本机的某个端口（本实验中是 2333），之后循环接收来自客户端的连接请求。接收到连接请求后，服务器建立一个新的线程用以与服务器进行交互。主线程继续监听，接收来自客户端的而请求。

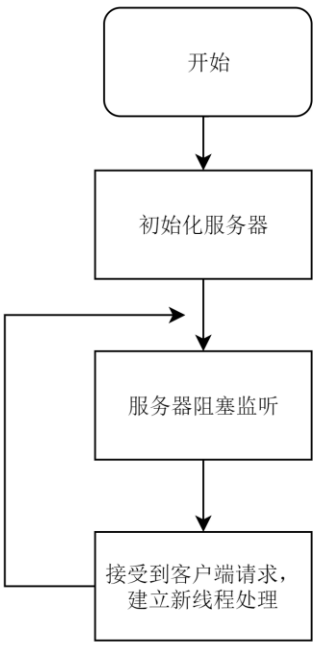

图 3-1 服务器端的执行流程

3.1.2 客户端的流程

客户端的流程如图 3-2 所示，首先会给用户一个主菜单，包括“connect to server”和“exit”两个选项。其中后者表示直接退出程序，“connect to server”选项则让用户输入服务器的 ip 和端口，并连接服务器。连接到服务器后，用户进入到功能菜单，有三个选项：（1）“download from server”，（2）“upload to server”，（3）“exit”。

其中最后一个选项用以返回上一级菜单，第（1）个选项客户端会向服务器发送下载文件的请求，第（2）个选项客户端会要求用户数据要上传的文件的路径，并且向服务器发送上传文件的请求。

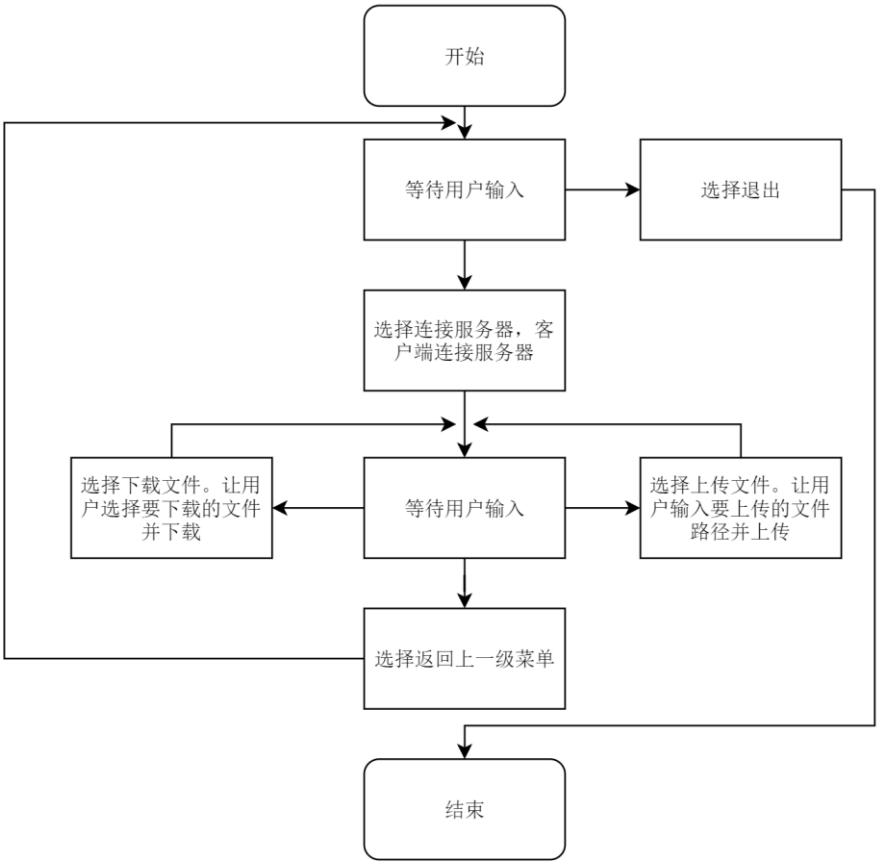

图 3-2 客户端的执行流程

3.2 端口扫描器的流程

由于端口扫描器的是一个 GUI 程序，用户可以在程序的执行过程中进行操作，图 3-3 给出了程序的一个主要的流程。该程序的 GUI 上提供三个按钮，请求清除 GUI 上的输出结果数据、开始扫描、停止扫描的功能。用户输入参数后点击开始扫描后，“开始扫描”按钮不再有效，只能操作其余两个按钮。也就是说，该程序只能同时存在一个扫描。

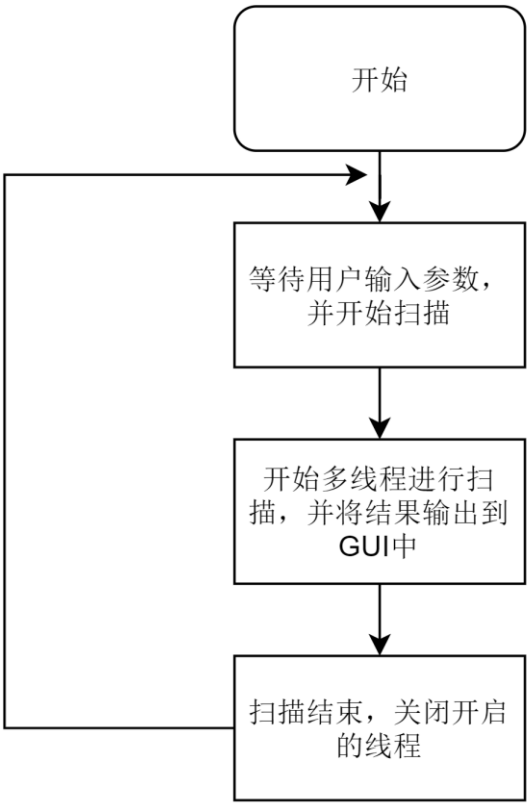

图 3-3 端口扫描器的主要执行流程

第 4 章 实现及结果

4.1 C/S 程序的实现

4.1.1 客户端实现

客户端在实现时有许多细节，我这里顺着程序执行的流程来描述客户端的实现。

首先，在主函数中，程序循环接收来自用户的输入。代码如图 4-1 所示。

```
int main(){
    int choice = -1;
    while(1){
        choice = main_menu();
        switch (choice)
        {
            case 1:
                client_handler();
                break;
            case 0:
                exit(0);
            default:
                printf("invalid input (0~1)!\n");
                break;
        }
    }

}
```

图 4-1 客户端的主函数

主函数的每次循环，程序首先调用函数 main_menu()，该函数向用户展示一个主菜单，并接收用户的输入。

如果用户选择连接服务器，主函数会调用函数 client_handler()，该函数会首先与服务器建立连接，然后向用户展示一个二级菜单。连接服务器的代码如图 4-2 所示。

```
if ((*sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("create socket error: %s(errno: %d)\n", strerror(errno),errno);
    return 0;
}

memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(port);
if (inet_pton(AF_INET, host, &servaddr.sin_addr)<=0) {
    printf("inet_pton error for %s\n", host);
    return 0;
}

if (connect(*sock, (struct sockaddr *)&servaddr, sizeof(servaddr))<0) {
    printf("connect error: %s(errno: %d)\n",strerror(errno),errno);
    return 0;
}
```

图 4-2 客户端连接服务器的代码

如果在连接服务器后，用户选择从服务器下载文件，程序会调用函数 client_recv()，该函数首先向服务器发送一个现在文件的请求，然后等待服务器的响应，服务器会返回可供下载的文件列表。客户端接收到列表后向用户展示，并向服务器请求用户选中的文件。此时，程序会调用 file_recv()函数，来接收文件，并存在文件中，保存时，程序会自动修改文件名称，以保证不与已存在的文件重名。

如果用户选择了上传文件到服务器，程序会调用函数 client_send()，该函数首先会让用户输入文件路径并检查文件是否存在，然后会发送一个上传文件的请求的请求给服务器，并在接收到服务器返回的确认消息后，调用函数 file_send()发送文件。

4.1.2 服务器端实现

服务器端的实现主要分为两部分，一个是主线程，用以循环接收客户端的连接请求；一个是具体与客户端交互的工作线程。

主线程的工作比较简单，首先初始化 socket，绑定服务器拥有的所有 ip 以及端口 2333，然后监听用户请求。初始化 socket 和绑定的代码如图 4-3 所示。

```c
// 创建套接字
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("create socket error: %s (errno: %d)\n", strerror(errno), errno);
    exit(1);
}

// 设置服务器端的端口以及 ip
memset(&serveraddr, 0, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
// serveraddr.sin_addr.s_addr = inet_addr("127.0.0.1");  // 设置 ip 为本机的 ip
serveraddr.sin_port = htons(SERVER_PORT);

// blind the address(port, ip, protocal) to the socket
if(bind(sock, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) == -1){
    printf("blind socket error: %s (errno: %d)\n", strerror(errno), errno);
    exit(1);
}

// start to listen to client's request
if (listen(sock, MAX_WAIT) == -1) {
    printf("listen socket error: %s (errno: %d)\n", strerror(errno), errno);
    exit(1);
}
```

图 4-3 服务器初始化 socket 并绑定端口、ip

监听相应端口后，主线程就会循环接收客户端的连接请求，并建立子线程与之交互。这里，新的线程依托函数 server_handler()函数与客户端交互。代码图 4-4 所示。

```c
while(1){
    if ((sock_c = accept(sock, (struct sockaddr *)&peeraddr, &len))<0){
        printf("accept error!\n");
        continue;
    }
    if (pthread_create(&thread_id, NULL, server_handler, (void *)(&sock_c)) == -1){
        printf("error occurs while creating new thread!\n");
        shutdown(sock_c, SHUT_RDWR);
    }else{
        getpeername(sock_c, (struct sockaddr *)&sa, &len);
        printf("accept a new client %s: %d\n", inet_ntoa(sa.sin_addr), ntohs(sa.sin_port));
    }
}
```

图 4-4 主线程为每个连接建立子线程

如果客户端发来下载文件的请求，服务器会调用 server_send()函数，该函数首先向客户端发送文件列表，然后根据客户端选定的某个文件，调用 file_send()函数来发送文件。这里的 file_send()函数与客户端使用的是同一个函数。

如果客户端发来上传文件的请求，服务器会调用 server_recv()函数，该函数会像客户端发送同意的消息，然后调用 file_recv()函数进行文件的接收，file_recv()与客户端使用同一个函数。

4.2 端口扫描器的实现

端口扫描器是一个 GUI 程序，可以分为前端和后端两部分。

对于前端界面部分，我使用 Python 自带的 GUI 包，tkinter。使用 Entry 控件来获取用户的输入，使用 Button 控件绑定后端的函数，用 Text 控件和 ScrollBar 控件来展示端口扫描的结果。GUI 上提供三个 Button，分别提供绑定 clear_data 函数来清除 Text 控件上的数据，绑定 start_scan_thread 来开启扫描，绑定 stop_scan 来停止扫描。

start_scan_thread 函数会检查是否正在进行扫描，如果没有则开启扫描，否则直接返回。代码如图 4-5 所示。

```
def start_scan_thread(inputs, texts):
    global scanner
    if scanner is not None and scanner.stopped() is False:
        texts.insert(tk.END, 'scanner is busy. Try again latter!\n')
        return
    scanner = scanthread(inputs, texts)
    scanner.start()
```

图 4-5 开启扫描

其中 scanthread 是一个继承自 threading.Thread 的类，该类中完成对输入数据的解析检查，并利用线程池进行扫描。在解析应该扫描的 IP 地址时，有一个小技巧，那就是将点分十进制的 IP 地址转换为一个整数，那么所有的 IP 就是一串连续的整数，可以轻松获得应该扫描的 IP，只需要将这些整数转换为点分十进制即可，这里由 addr2dec（如图 4-6）和 dec2addr 函数（如图 4-7）来完成这个功能。

```
def addr2dec(addr):
    """
    将点分十进制转换为整数
    """
    items = [int(x) for x in addr.split('.')]
    return sum([items[i] << [24, 16, 8, 0][i] for i in range(4)])
```

图 4-6 将点分十进制形式的 IP 转换为整数

```
def dec2addr(dec):
    """
    将十进制IP转换为点分十进制
    """
    return '.'.join([str(dec>>x & 0xff) for x in [24, 16, 8, 0]])
```

图 4-7 将整数形式的 IP 转换为点分十进制

扫描器利用线程池进行多线程扫描的代码如图 4-8 所示：

```
reqs = threadpool.makeRequests(scan_thread, urls_list)
for req in reqs:
    self._pool.putRequest(req)
try:
    self._pool.joinAllDismissedWorkers()
    self._pool.wait()
except threadpool.NoWorkersAvailable:
    print('scan stopped...')
```

图 4-8 利用线程池进行多线程扫描

其中，每个线程依托的函数是 scan_thread，其实现如图 4-9 所示。

```
def scan_thread(urls):
    s = socket.socket()
    texts=urls[0]

    for i in range(1, len(urls), 1):
        host, port = urls[i]
        try:
            s.connect(urls[i])
            texts.insert(tk.END, host+': '+str(port)+'...open\n')
        except socket.error:
            texts.insert(tk.END, host+': '+str(port)+'...closed\n')
    s.close()
```

图 4-9 scan_thread 函数的实现

可以看到，每得到一个扫描结果，函数都会调用前端的 Text 控件的 insert 函数，将结果显示出来。

4.3 实验结果

4.3.1 C/S 程序的结果

客户端的主菜单如图 4-10 所示。

```
|-----------------------------------------|
|                  menu                   |
|   1. connect to server                  |
|   0. exit                               |
|-----------------------------------------|
enter your choice:
```

图 4-10 客户端的主菜单

二级菜单如图 4-11 所示。

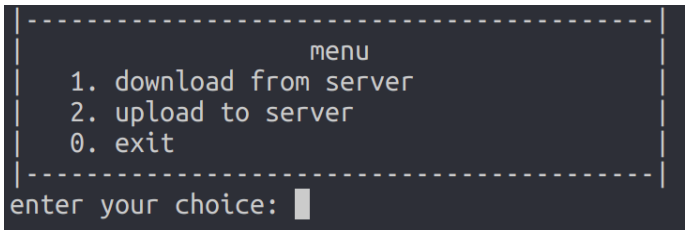

图 4-11 二级菜单

由于服务器和客户端使用同一套函数 file_send()和 file_recv()就行文件的收发，因此这里仅给出客户端接收文件的结果。其中，客户端和服务器接收到文件都会存入 data 目录下，图 4-12 是接收文件前 data 目录下的文件，图 4-13 是接收文件后 data 目录下的文件。

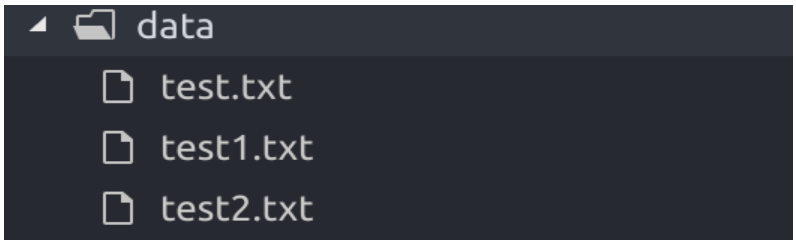

图 4-12 接收文件前 data 目录下的文件

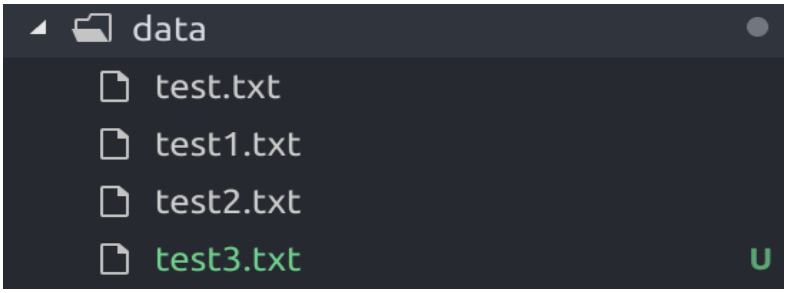

图 4-13 接收文件后 data 目录下的文件

经过验证，两个文件是完全相同的。

4.3.2 端口扫描器的结果

扫描百度服务器的前 500 个端口，可以得到如图 4-10 的结果。

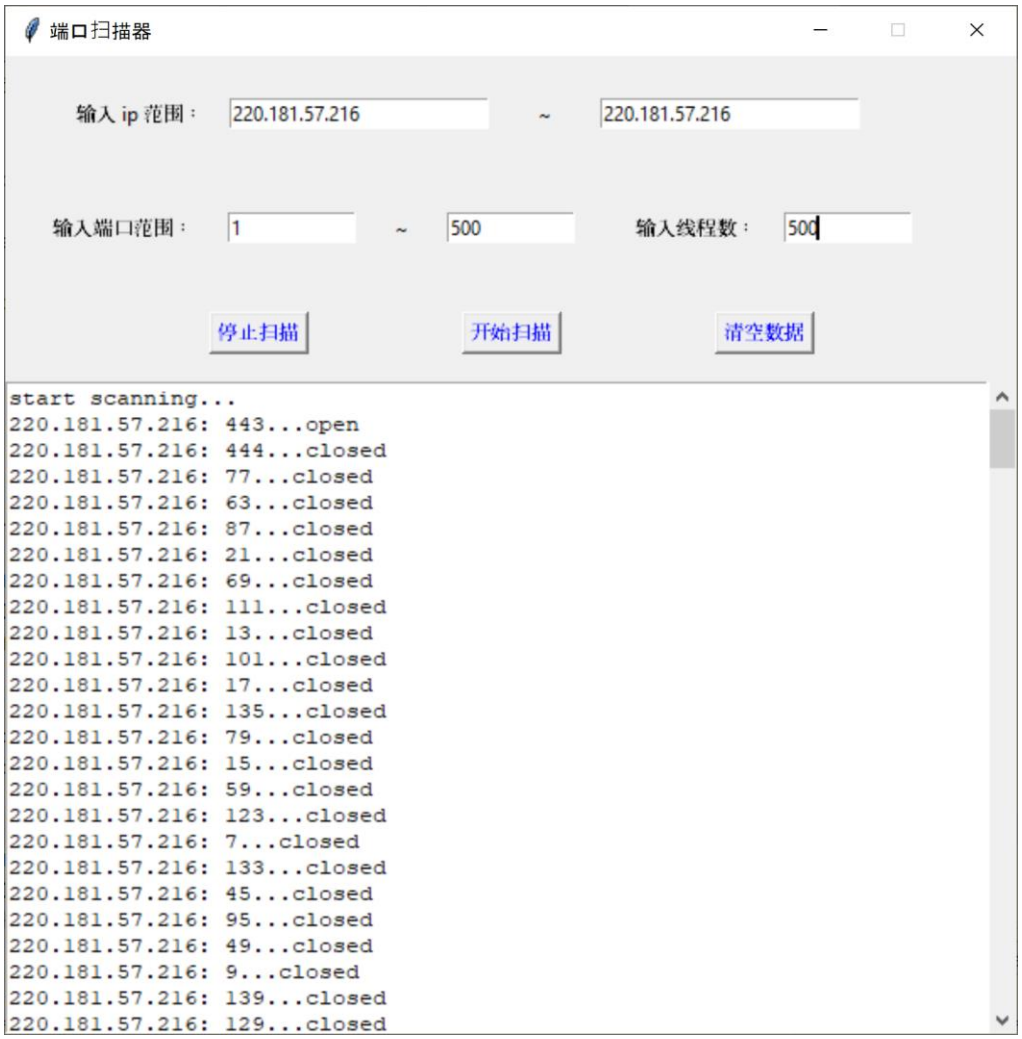

图 4-10 扫描百度服务器的前 500 个端口的结果

第 5 章 总结

5.1 本次实验的收获

（1）熟悉了在 Linux 环境下使用 C 语言进行编程，尤其是 Linux 下的 socket 编程；
（2）了解了端口扫描的基本原理，并能动手实现；
（3）了解了 Python 下的 GUI 编程；

5.2 对本次实验的建议

虽然目的就是为了不给实验内容过多的约束，但是我觉得实验指导给得还是不够充分，比如对实验的环境要求并没有表述清楚。指导中并没有写明编程语言的要求，其中第一部分却要求使用 C 语言。

第 6 章 参考文献

[1] http://www.linuxhowtos.org/C_C++/socket.htm.