# Wireshark入门

|  |  |
| --- | --- |
| 实验名称：Wireshark入门 | |
| 实验台号：2 | 实验时间：2019年11月12日 |
| 实验小组：第2组  成员及本次实验分工:於文卓 | |
| 实验目的：  1.Wireshark的基本介绍，包括显示界面、开始界面等。  2.了解Wireshark的相关设置，包括首选项设置，抓包选项设置，过滤器设置等。 | |
| 实验环境说明：  笔记本电脑一台(Win10系统)，Wireshark(Version 3.0.3)。 | |
| 实验过程、步骤（可另附页、使用网络拓扑图等辅助说明）及结果：  1.什么是Wireshark？  Wireshark（前称Ethereal）是一个网络数据包分析软件。网络数据包分析软件的功能是截取网络数据包，并尽可能显示出最为详细的网络数据包数据。Wireshark使用WinPCAP作为接口，直接与网卡进行数据报文交换。  网络管理员使用Wireshark来检测网络问题，网络安全工程师使用Wireshark来检查资讯安全相关问题，开发者使用Wireshark来为新的通讯协定除错，普通使用者使用Wireshark来学习网络协定的相关知识。Wireshark相对于tcpdump而言，界面更友好，功能更强大。  2.界面认识  欢迎界面如下。Wireshark是捕获机器上的某一块网卡的网络包，当你的机器上有多块网卡的时候，需要选择一个网卡。    2.1菜单栏常用功能，从左到右依次是开始捕获分组、停止捕获分组、重新开始捕获分组。打开、保存、关闭、重新加载文件。和分组的查找以及转换。    2.2过滤器主要有两种，一种是显示过滤器，就是主界面上的那个。用来在捕获的记录中找到所需要的记录。还有一种是捕获过滤器，用来过滤捕获的封包，以免捕获太多的记录。在捕获->捕获过滤器中设置。    2.3封包列表中展示的是获取的数据包，面板中显示了编号、时间戳、原地址、目标地址、协议、长度以及封包信息等。也可以修改不同规则的颜色。      3.数据包的保存  完成数据包的捕获后，可能我们并不急着马上做分析，或者说当前能做的分析还不够完整，需要后面来加深……如此种种，我们需要用文件保存这些数据包。保存数据包也有三种方式：  1.使用Ctrl+S组合键。  2.菜单栏：依次点击"File"-->"Save"。  3.主工具栏的按钮。    4.首选项中的一些配置  4.1调整界面位置  可以调整抓包页面的布置，这根据个人喜好，不过以经验来说还是默认的最好用。    4.2捕获窗口的一些设置  第一个和第二个选项使得转包的时候前端实时滚动显示抓包详情。第三个选项默认是隐藏转包协议比例显示，比较耗费资源。MAC地址解析会自动将MAC地址转化成易识别设备名。第五第六个如果勾选，会自动将重要ip地址转化成域名，方便识别。 | |
| 实验总结（遇到的问题及解决办法、体会）：  本实验主要是带我们了解了Wireshark的入门。Wireshark是非常流行的网络封包分析软件，功能十分强大。可以截取各种网络封包，显示网络封包的详细信息。使用Wireshark的人必须了解网络协议，否则就看不懂Wireshark。为了安全考虑，Wireshark只能查看封包，而不能修改封包的内容，或者发送封包。Wireshark能获取http，也能获取https，但是不能解密HTTPS，所以Wireshark都不懂HTTPS中的内容。  总结，如果是处理HTTP，或者其他协议比如TCP，UDP就用Wireshark。  Wireshark中的过滤是非常重要的，能帮助我们快速在上千条记录中快速找到自己需要的部分。 | |
| 器材、工具领用及归还负责人：於文卓 | 实验记录人：於文卓 |
| 实验执笔人：於文卓 | 报告协助人：於文卓 |
| 小组成员签名：於文卓 | |
| 验收人： | 成绩评定： |

# 基本报文分析

|  |  |
| --- | --- |
| 实验名称：基本报文分析 | |
| 实验台号：2 | 实验时间：2019年11月17日 |
| 实验小组：第2组  成员及本次实验分工:於文卓 | |
| 实验目的：  1.理解IP层的作用以及IP地址的分类方法。  2.理解子网的划分和子网掩码的作用。  3.掌握IP数据包的组成和网络层的基本功能。  4.学习使用Wireshark如何对报文作分析。  5.通过Wireshark了解TCP三次握手过程。 | |
| 实验环境说明：  笔记本电脑一台(Win10系统)，Wireshark(Version 3.0.3) | |
| 实验过程、步骤（可另附页、使用网络拓扑图等辅助说明）及结果：  1.原理分析  网际协议 IP 是 TCP/IP 协议栈的心脏，也是网络层中最重要的协议。目前几乎所有的上层网络协议都是基于 IP 协议。在接收数据的时候，网络层接收由数据链路层发送的数据包进行解封装，并把该数据包发送到更高层——传输层，在发送数据的时候，网络层接受由传输层发送的数据包进行 IP 封装，然后把数据报交给下层——数据链路层。  IP 协议处于 TCP/IP 协议栈的网际层，用于管理数据通信中源端和目的端之间的报文传送，是互联网最重要的网际协议。IP 地址是也叫逻辑地址，用于在网络中标识主机。在 IP网络中，主机之间进行通信时使用 IP 地址来指定接收端的主机地址。数据进行封装过程中，IP层负责将数据封装成 IP包，IPv4数据包报文格式如下图所示。    2.Wireshark抓包对应的OSI模型    3.Wireshark抓到的包对应的TCP中的每个字段    4.实例分析TCP三次握手  首先打开Wireshark，再在浏览器中输入www.hdu.edu.cn 在Wireshark中进行http过滤后，选中GET / HTTP/1.1的那条记录，右键然后点击追踪流🡪TCP流，从而得到与浏览器打开网站相关的数据包    从图中可以看到Wireshark截获了三次握手的三个数据包。第四个包才是HTTP，说明HTTP是使用TCP建立连接的。    第一次握手：  客户端发起一个TCP，标志位为SYN，序列号为0，代表客户端请求连接。    第二次握手：  服务器发回确认包，标志位为SYN，ACK，将确认序号设置为客户端的ISN+1。    第三次握手  客户端再次发送确认包（ACK），SYN标志位为0，ACK标志位为1，并且把服务器发来ACK的序号字段+1，放在确定字段中发给对方，并且数据字段放些ISN的+1。 | |
| 实验总结（遇到的问题及解决办法、体会）：  问题：IP 数据包在从源主机出发到达目的主机的过程中，IP首部中的I源地址和目的地址字段是否发生变化？  回答：没有变化。在数据封装的过程中 是包含协议端口等其他标识应用程序的字段或者协议，以便应用层能够区分，在internet层（OSI中的网络层）完成对IP地址的封装之后数据交到数据链路层 打上帧头 帧尾 之后交下一层进行传输在传输过程中不变化 | |
| 器材、工具领用及归还负责人：於文卓 | 实验记录人：於文卓 |
| 实验执笔人：於文卓 | 报告协助人：於文卓 |
| 小组成员签名：於文卓 | |
| 验收人： | 成绩评定： |

# DNS域名服务协议

|  |  |
| --- | --- |
| 实验名称：DNS域名服务协议 | |
| 实验台号：2 | 实验时间：2019年11月22日 |
| 实验小组：第2组  成员及本次实验分工:於文卓 | |
| 实验目的：  1.理解DNS实现的原理和实现方法。  2.了解DNS解析的过程。  3.通过编辑DNS请求数据包，了解DNS的报文格式。  4.掌握nslookup命令和ipconfig命令的使用方法。  5.通过dig命令可以显示整个DNS查询过程。 | |
| 实验环境说明：  笔记本电脑一台(Win10系统)，Wireshark(Version 3.0.3) | |
| 实验过程、步骤（可另附页、使用网络拓扑图等辅助说明）及结果：  1.原理分析  DNS 域名系统是服务器和客户程序相互通信的一种协议。它提供了主机域名和 IP 地址之间的转换。域名服务器使用固定的端口号53，支持 UDP 和 TCP 访问。  DNS 是域名系统（Domain Name System）的缩写，它是一种用于 TCP/IP 应用程序的分布式数据库，它提供主机名字和 I P 地址之间的转换及有关电子邮件的选路信息。所谓“分布式”是指在 Internet 上的单个站点不能拥有所有的信息。每个站点（如大学中的系、校园、公司或公司中的部门）保留它自己的信息数据库，并运行一个服务器程序供 Internet上的其他系统（客户程序）查询。  域名解析需要由专门的域名解析服务器来完成，DNS 就是进行域名解析的服务器。它是一种分布式网络目录服务，主要用于域名与 IP 地址的相互转换，以及控制因特网的电子邮件的发送。大多数因特网服务依赖于 DNS 而工作，一旦DNS 出错，就无法连接 Web 站点，电子邮件的发送也会中止。  如图是DNS的解析流程。[图片来源](http://www.linuxdiyf.com/linux/21226.html)    2.实验步骤  2.1 步骤一：使用Wireshark捕获数据包并进行分析  在Wireshark的工具栏输入框输入DNS即可筛选出DNS协议的记录。    通过Wireshark的抓包，可以看到DNS报文的具体内容，如图所示。    2.2 步骤二：使用nslookup工具解析域名  nslookup命令是查询域名对应ip的工具，其用法可以直接在windows系统的cmd中运行命令：nslookup 域名 来进行域名解析。    也可以仅仅运行nslookup命令，不加任何参数。进入nslookup的交互界面，在“>”提示符后可以多次输入不同的域名，以实现多次的查询。如图实现了对[www.baidu.com、www.sina.com、www.hdu.edu.cn](http://www.baidu.com、www.sina.com、www.hdu.edu.cn)的查询。    最后使用exit命令退出nslookup。  2.3 步骤三：使用ipconfig命令查看DNS缓存  在Windows系统的cmd中，运行 ipconfig /displaydns 显示本机缓存区中的DNS解析内容。    而使用ipconfig /flushdns，可以清楚本机的DNS缓存记录。  2.4 使用dig工具显示DNS查询过程  在Windows中使用dig命令需要安装相应程序，可以直接在网页查询，网址<https://www.diggui.com/>  在网页中可以选择要查询的类型、域名，选择公共的DNS服务器等。    对[www.baidu.com](http://www.baidu.com)的查询结果如下。    可以看到[www.baidu.com](http://www.baidu.com)对应一个别名[www.a.shifen.com](http://www.a.shifen.com)，而这个域名又对应一个别名[www.wshifen.com](http://www.wshifen.com)。通过查询这个域名的ip地址，我们最终得到了两个ip，这两个ip为[www.baidu.com](http://www.baidu.com)域名对应的ip，在浏览器中输入这两个ip即可访问[www.baidu.com](http://www.baidu.com)。注意到，[www.baidu.com](http://www.baidu.com)后面还有个“.”，这是为什么呢。这并不是疏忽，其实所有域名的尾部，实际上都有一个根域名。比如[www.example.com](http://www.example.com)的真正域名其实是[www.baidu.com.root，之所以简写，是因为.root](http://www.baidu.com.root，之所以简写，是因为.root)对于所有域名都是一样的。    3.其他工具介绍  3.1 host命令  Host命令可以看做dig命令的简化版本，返回当前请求域名的各种记录  3.2 whois命令  Whois命令用来查看域名的注册情况 | |
| 实验总结（遇到的问题及解决办法、体会）：  通过这次实验，我进一步了解了DNS的作用和工作流程。额外的通过Wireshark抓包，了解了DNS报文的结构。同时还了解了Windows中关于DNS的一些命令比如nslookup、ipconfig等。通过nslookup，我可以直接查询域名对应的IP地址，而通过ipconfig命令，我可以知道本地缓存了那些DNS解析内容。  此外了解使用了dig命令，可以显示DNS的查询过程。  域名服务分为客户端和服务器端，客户端提出请求，询问一个 Domain Name 的 IP 地址，服务器端必须回答客户端的请求。本地 DNS 首先查询自己的数据库，如果自己的数据库中没有对应的 IP 地址，则向本地 DNS 上所设的上一级 DNS 询问，得到结果之后，将收到的结果保存在高速缓冲区，并回答给客户端。  问题：域名与 IP 地址之间是否有一一对应的关系？  IP对域名是一对多的关系，一个IP地址可以对应多个域名；一个域名只能解析一个IP地址。 | |
| 器材、工具领用及归还负责人：於文卓 | 实验记录人：於文卓 |
| 实验执笔人：於文卓 | 报告协助人：於文卓 |
| 小组成员签名：於文卓 | |
| 验收人： | 成绩评定： |

# TCP网络程序设计

|  |  |
| --- | --- |
| 实验名称：TCP网络程序设计 | |
| 实验台号：2 | 实验时间：2019年11月24日 |
| 实验小组：第2组  成员及本次实验分工:於文卓 | |
| 实验目的：  1.进一步学习TCP协议的工作原理。  2.学习SOCKET编程的基本方法。  3.使用Python编程语言的socket包，实现客户端对服务端的一对一和多对多数据传输。 | |
| 实验环境说明：  笔记本电脑一台(Win10系统)，Wireshark(Version 3.0.3)，Python（3.6）。 | |
| 实验过程、步骤（可另附页、使用网络拓扑图等辅助说明）及结果：  1.相关概念简介  1.1 套接字（socket）  IP地址加端口号的方式就构成了网络通信过程中的唯一标识符，即套接字socket。通过套接字(ip + port)可以实现多主机多应用程序间同时通信，极大地提高了网络的应用能力。常用的socket类型有两种：流式Socket（SOCK\_STREAM）和数据报式Socket（SOCK\_DGRAM）。流式是一种面向连接的Socket，针对于面向连接的TCP 服务应用；数据报式Socket 是一种无连接的Socket，对应于无连接的UDP 服务应用。  1.2 TCP  TCP传输控制协议，提供的是面向连接、可靠的字节流服务。当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP 连接的请求。  2.实验步骤    2.1 相关模块介绍  1）创建套接字   |  | | --- | | from socket import \*  # AF\_INET 采用ipv4的tcp或udp通信机制  # SOCK\_STREAM代表tcp通信  sockobj = socket(AF\_INET， SOCK\_STREAM) |   2）绑定ip和端口   |  | | --- | | sockobj.bind((myHost， myPort)) |   3）监听  Python的tcp服务器需要调用socket模块里的listen函数来等待客户端连接，listen函数的参数可以设置服务器可同时接收多少个客户端同时在线。   |  | | --- | | sockobj.listen(5) |   4）接受连接   |  | | --- | | connection， address = sockobj.accept() |   第一个返回值是一个通信套接字，第二个返回值是客户的ip和端口信息。  5）发送数据   |  | | --- | | # 向客户端发送或者从客户端接受数据  connection.send()  data = connection.recv() |   6）关闭请求套接字，关闭服务套接字   |  | | --- | | # 关闭  connection.close()  sockobj.close() |   2.2 实现简单的一对一TCP通信程序  1）server.py 服务端代码  在服务端中，设置ip为localhost，端口号为50007。当服务端收到客户端传来的消息时，打印消息，并向客户端传送：Got it。   |  | | --- | | from socket import \*  #""表示localhost  myHost = ""  myPort = 50007  # 设置一个TCP socket对象  sockobj = socket(AF\_INET， SOCK\_STREAM)  # 绑定端口号  sockobj.bind((myHost， myPort))  # 监听，允许5个连结  sockobj.listen(5)  # 等待客户端连接  connection， address = sockobj.accept()  print("Server conneted by"， address)  data = connection.recv(1024)  print("Server received data:"， data.decode('utf-8'))  connection.send("Echo from server:Got it!".encode("utf-8"))  connection.close()  sockobj.close() |   2）客户端代码   |  | | --- | | from socket import \*  from time import sleep  serverHost = 'localhost'  serverPort = 50007  # 建立一个tcp/ip套接字对象  sockobj = socket(AF\_INET， SOCK\_STREAM)  # 连接至服务器及端口  print("正在连接服务器")  sleep(1)  sockobj.connect((serverHost， serverPort))  print("连接成功")  message = input("请输入想发送的消息：")  sockobj.send(message.encode('utf-8'))  print("Client sent:"， message)  # 从服务端接收到的数据，上限为1k  data = sockobj.recv(1024)  print('Client received:'， data.decode('utf-8'))  sockobj.close() |   3）实现结果    可以看到，服务端通过listen监听到了客户端的连接，并收到了客户端发来的消息hello world。客户端发送消息后，收到了来自服务端的Got it！说明TCP建立成功。  2.3 实现多对多TCP通信程序  通常一个服务器端程序在计算机里是一个进程，如果有多个客户端同时上线要和服务器程序进行多对多的通信，需要服务器端软件创建若干个线程，由进程创建的线程和每一个客户端进行一对一的通信，交换数据。如果服务器端的程序没有创建线程，那无论有多少个客户端程序向服务器端发起请求连接，也仅有一个客户端被服务，因为服务器端就一个进程，不可能一下对付那么多的通信请求。  所以如果需要实现TCP的多对多通信，需要使用多线程。  基本思想是每次有客户端请求服务的时候，服务器端的程序就创建一个新的线程专门服务于该客户端的服务请求。下面的server.py代码有使用了threading模块来创建线程的内容。   1. server.py  |  | | --- | | from socket import \*  import threading  def stoc(client\_socket， addr):  while True:  try:  client\_socket.settimeout(500)  buf = client\_socket.recv(1024)  print("\*" \* 10)  print("msg from:"， addr[1])  print("msg:"， buf.decode('utf-8'))  print("\*" \* 10)  except socket.timeout:  print("Time Out")  break  #""表示localhost  myHost = ""  myPort = 50007  # 设置一个TCP socket对象  server = socket(AF\_INET， SOCK\_STREAM)  # 绑定端口号  server.bind((myHost， myPort))  # 监听，允许5个连结  server.listen(5)  while True:  # 等待客户端连接  client， address = server.accept()  print("Server conneted by"， address)  thread = threading.Thread(target=stoc， args=(client， address))  thread.start() |   2）client.py  客户端的代码和原先的基本一致   |  | | --- | | from socket import \*  from time import sleep  serverHost = 'localhost'  serverPort = 50007  # 建立一个tcp/ip套接字对象  client = socket(AF\_INET， SOCK\_STREAM)  # 连接至服务器及端口  print("正在连接服务器")  sleep(1)  client.connect((serverHost， serverPort))  print("连接成功")  message = input("请输入想发送的消息：")  client.send(message.encode('utf-8'))  print("Client sent:"， message)  # 从服务端接收到的数据，上限为1k  data = client.recv(1024)  print('Client received:'， data.decode('utf-8'))  client.close() |   3）实验结果  可以看到，本实验开启了一个服务端和三个客户端，三个客户端连接同一个服务端后，分别发送不同的信息。可以看到服务端收到来自三个不同客户端的信息，并能分辨来自哪个客户端。实现了TCP的多对多通信。 | |
| 实验总结（遇到的问题及解决办法、体会）：  TCP传输控制协议，提供的是面向连接、可靠的字节流服务。通过本次实验，进一步了解了TCP协议的工作方式及其原理。使用Python编程语言的socket包，实现客户端对服务端的一对一的传输。但由于服务器进程只有一个，如果需要实现多对多的通信，需要使用python的Threading库，通过这个库，通过对多线程的使用，最终实现了TCP的多对多通信。  问题：  1. 根据编程练习实验中记录的客户和服务器程序的端口号并结合程序，说明：在客户/服务器模型当中，客户进程的端口号和服务器进程的端口号都是由程序给出说明的吗？为什么？  答：是的，客户端程序通过指定的端口号来寻找相应的服务进程。  2. 在 TCP/IP 网络中，当客户与服务器进程建立了一条 TCP 连接以后，是否属于该连 接的所有包都是经过同一路径（即一条虚电路）传递的？为什么？  答：是的；虚电路建立后，通信双方就沿着已建立的虚电路发送分组。这样首部不需要填写完整的目的主机地址，只需填写这条虚电路的编号。 | |
| 器材、工具领用及归还负责人：於文卓 | 实验记录人：於文卓 |
| 实验执笔人：於文卓 | 报告协助人：於文卓 |
| 小组成员签名：於文卓 | |
| 验收人： | 成绩评定： |

# UDP网络程序设计

|  |  |
| --- | --- |
| 实验名称：UDP网络程序设计 | |
| 实验台号：2 | 实验时间：2019年11月24日 |
| 实验小组：第2组  成员及本次实验分工:於文卓 | |
| 实验目的：  1.进一步学习UDP协议的工作原理。  2.学习SOCKET编程的基本方法。  3.使用Python编程语言的socket包，实现客户端对服务端的数据传输。 | |
| 实验环境说明：  笔记本电脑一台(Win10系统)，Wireshark(Version 3.0.3)，Python（3.6） | |
| 实验过程、步骤（可另附页、使用网络拓扑图等辅助说明）及结果：  1.相关概念简介  1.1 UDP  UDP用户数据报协议，是一个无连接的简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP 在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快。  2.实验步骤   2.1 相关模块介绍 1）创建套接字  和Tcp一样用socket.socket来创建服务套接字，指定套机字的类型和通信类型socket.SOCK\_DGRAM代表是采用udp方式的套接字通信。   |  | | --- | | import socket  server\_socket = socket.socket(AF\_INET， SOCK\_DGRAM) |   2）绑定ip和端口  基于套接字通过bind函数对外公布地址和端口号   |  | | --- | | sockobj.bind((myHost， myPort)) |   3）接受数据  udp收数据用recvfrom函数   |  | | --- | | data， addr = server\_socket.recvfrom(字节数) |   recvfrom的返回值有两个，第一个是收到的数据，第二个是客户端套接字(地址和端口号)，即从那个客户端收到了数据。  4）发送数据  这里的addr是某客户端的套接字(地址和端口号)，可以直接指出也可使用recvfrom的第二个返回值   |  | | --- | | server\_socket.sendto(数据，addr) |   5）关闭服务套接字   |  | | --- | | server\_socket.close() |   2.2 实现UDP多对多数据传输  1）server.py 服务端代码  在服务端中，设置ip为localhost，端口号为50007。当服务端收到客户端传来的消息时，打印消息，并向客户端传送：Got it。   |  | | --- | | from socket import \*  from time import sleep， ctime  host = ""  port = 50007  addr = (host， port)  server = socket(AF\_INET， SOCK\_DGRAM)  server.bind(addr)  print("服务器建立成功")  while True:  data， address = server.recvfrom(1024)  print("从{}中获得数据{}".format(address， data.decode()))  server.sendto("Got it!".encode()， address)  print("\*" \* 20) |   2）客户端代码  客户端想服务器发送字用户输入的数据，发送出成功后，得到服务器的回复Got it！   |  | | --- | | from socket import \*  from time import ctime， sleep  host = "localhost"  port = 50007  addr = (host， port)  client = socket(AF\_INET， SOCK\_DGRAM)  while True:  msg = input("输入要发送的消息:")  print("正在发送...")  sleep(1)  client.sendto(msg.encode()， addr)  print("发送成功")  data， address = client.recvfrom(1024)  print("Server: "， data.decode())  print("\*" \* 20) |   3）实现结果    可以看到创建了一个服务端和三个客户端，服务端收到了客户端发来的消息。客户端发送消息后，收到了来自服务端的Got it！说明UDP通讯建立成功。 | |
| 实验总结（遇到的问题及解决办法、体会）：  TCP VS UDP  1) TCP  a.基于链接，则需要listen（backlog），指定连接池的大小。  b.基于链接，必须先运行的服务端，然后客户端发起链接请求。  c.如果一端断开了链接，那另外一端的链接也跟着断开(在mac系统中，recv收到的是空，而在windows和linux中，会直接报错，需要在代码中加入异常处理)。  2)UDP  a.无链接，因而无需listen（backlog）  b.无链接，udp的sendinto不用管是否有一个正在运行的服务端，可以己端可以不停地发消息，只不过数据丢失。如图所示，并没有开启服务器，但客户端依然可以发送数据。 | |
| 器材、工具领用及归还负责人：於文卓 | 实验记录人：於文卓 |
| 实验执笔人：於文卓 | 报告协助人：於文卓 |
| 小组成员签名：於文卓 | |
| 验收人： | 成绩评定： |

# 基于Python Socket的聊天程序

|  |  |
| --- | --- |
| 实验名称：基于Python Socket的聊天程序 | |
| 实验台号：2 | 实验时间：2019年11月29日 |
| 实验小组：第2组  成员及本次实验分工:於文卓 | |
| 实验目的：  1.进一步学习TCP协议的工作原理。  2.学习SOCKET编程的基本方法。  3.使用Python编程语言的socket包，实现多个服务器客户端之间一对多的通信。  4.使用Tkinter包，实现一个聊天应用程序界面。 | |
| 实验环境说明：  笔记本电脑一台(Win10系统)，Python（3.6）。 | |
| 实验过程、步骤（可另附页、使用网络拓扑图等辅助说明）及结果：  1.引言  在这个实验中，我使用Python和Python中的Socket包，实现了一个基于TCP的聊天小程序，在实现过程中使用了多线程技术，使得一个服务端能同时和多个客户端交流，客户端之间也能同时接受和发送消息。此外还是用Tkinter库，实现了一个GUI界面，相较于之前的控制台程序，GUI界面的程序使用起来更加方便。以下是实现过程介绍。  2.实验步骤    2.1 服务端实现  1）导入所需要的库  AF\_INET 采用ipv4的tcp或udp通信机制，我的小程序是基于ipv4实现的。SOCK\_STREAM代表tcp通信，我顺便了解了微信的文本交流使用的是TCP，而QQ更多的是使用UDP，我的小程序基于的是TCP的协议。   |  | | --- | | #!/usr/bin/env python3  from socket import AF\_INET， socket， SOCK\_STREAM  #导入多线程库Thread from threading import Thread |   2）设定一些常量  定义了服务端的一些基本配置，比如主机地址和端口号，初始化了socket对象，并把主机地址和端口号绑定到了这个socket上。另外还设置了两个字典，client字典用来存放每个客户的名字，addresses字典用来保存每个客户端的地址信息。   |  | | --- | | clients = {} addresses = {}  HOST = '' PORT = 33000 BUFSIZ = 1024 ADDR = (HOST， PORT) SERVER = socket(AF\_INET， SOCK\_STREAM) SERVER.bind(ADDR) |   3）等待连接  使用socket中的accept函数，服务器等待客户端的连接。这个函数有两个返回值，一个是用来通信的socket，一个是客户端的地址。连接成功后，服务器会告诉客户端，可以在输入框中输入信息，提供自己的名字。   |  | | --- | | def accept\_incoming\_connections():  while True:  client， client\_address = SERVER.accept()  print("%s:%s has connected." % client\_address)  client.send(bytes("Now type your name and press enter!"， "utf8"))  addresses[client] = client\_address  Thread(target=handle\_client， args=(client，)).start() |   4）处理客户端  这个函数用来处理客户端的相关操作。比如使用recv函数接受来自客户端的消息。用send函数对客户端发送消息。如果客户端发送了{quit}这样的消息，那么说明客户端退出了，需要关闭相关内容，包括用来通信的socket，要删除字典中对应客户端的信息等，并广播说明，这个客户端退出了。   |  | | --- | | def handle\_client(client):  name = client.recv(BUFSIZ).decode("utf8")  welcome = 'Welcome %s! If you ever want to quit， type {quit} to exit.' % name  client.send(bytes(welcome， "utf8"))  msg = "%s has joined the chat!" % name  broadcast(bytes(msg， "utf8"))  clients[client] = name  while True:  msg = client.recv(BUFSIZ)  if msg != bytes("{quit}"， "utf8"):  broadcast(msg， name+": ")  else:  client.send(bytes("{quit}"， "utf8"))  client.close()  del clients[client]  broadcast(bytes("%s has left the chat." % name， "utf8"))  break |   5）广播  广播顾名思义就是要对所有的客户端都发送同样的信息，这里遍历字典中保存的所有在线的客户端，发送同样的信息。在很多及时通讯软件中，有群组的概念。最简单的群消息处理就是讲一个客户端的消息传到中心服务器后，中心服务器将这个消息发给这个群里的每个人。   |  | | --- | | def broadcast(msg， prefix=""):  for sock in clients:  sock.send(bytes(prefix， "utf8")+msg) |   6）开启服务端  我的服务器实现了基本操作后，就可以开启等待客户端的访问了。这里listen表示服务器最多能被多少个客户端访问。使用Thread可以开启多线程，即当有新的客户端对这个服务器进行访问时，就开启一个线程处理这个客户端。   |  | | --- | | if \_\_name\_\_ == "\_\_main\_\_":  SERVER.listen(5)  print("Waiting for connection...")  ACCEPT\_THREAD = Thread(target=accept\_incoming\_connections)  ACCEPT\_THREAD.start()  ACCEPT\_THREAD.join()  SERVER.close() |   2.2 客户端实现  1）导入所需要的库  与服务端的操作类似，这里导入课tkinter，python自带的一个GUI图形界面包。   |  | | --- | | #!/usr/bin/env python3  from socket import AF\_INET， socket， SOCK\_STREAM from threading import Thread import tkinter |   2）接收消息  客户端需要从服务端接受消息，使用recv函数，获得来自服务端的消息后插入到消息列表中，这个消息列表的信息会展示在图形界面的消息框中。   |  | | --- | | def receive():  while True:  try:  msg = client\_socket.recv(BUFSIZ).decode("utf8")  msg\_list.insert(tkinter.END， msg)  except OSError: # Possibly client has left the chat.  break |   3）发送消息  客户端通过图形界面，通过send函数将消息发送给服务端。msg消息是通过my\_msg.get()函数得到的，这里的my\_msg是一个消息输入框。如果发送的是{quit}，需要退出这个客户端。   |  | | --- | | def send(event=None):  msg = my\_msg.get()  my\_msg.set("") # Clears input field.  client\_socket.send(bytes(msg， "utf8"))  if msg == "{quit}":  client\_socket.close()  top.quit() |   4）Tkinter图形界面相关代码  这里设置了窗口的名字，大小。设置了相关组件包括Scrollbar，Listbox等，还有一些处理细节比如在输入框中敲回车，也可以激活发送命令，关比客户端窗口，会触发close命令等。   |  | | --- | | top = tkinter.Tk()  top.title("Chatter")  messages\_frame = tkinter.Frame(top)  my\_msg = tkinter.StringVar() # For the messages to be sent.  my\_msg.set("Type your messages here.")  scrollbar = tkinter.Scrollbar(messages\_frame)  msg\_list=tkinter.Listbox(messages\_frame，height=15，width=50，yscrollcommand=scrollbar.set)  scrollbar.pack(side=tkinter.RIGHT， fill=tkinter.Y)  msg\_list.pack(side=tkinter.LEFT， fill=tkinter.BOTH)  msg\_list.pack()  messages\_frame.pack()  entry\_field = tkinter.Entry(top， textvariable=my\_msg)  entry\_field.bind("<Return>"， send)  entry\_field.pack()  send\_button = tkinter.Button(top， text="Send"， command=send)  send\_button.pack() |   5）设置客户端信息  设置客户端的端口、主机号等信息，初始化客户端的socket，开启多线程，开启tkinter。   |  | | --- | | HOST = input('Enter host: ')  PORT = input('Enter port: ')  if not PORT:  PORT = 33000  else:  PORT = int(PORT)  BUFSIZ = 1024  ADDR = (HOST， PORT)  client\_socket = socket(AF\_INET， SOCK\_STREAM)  client\_socket.connect(ADDR)  receive\_thread = Thread(target=receive)  receive\_thread.start()  tkinter.mainloop() |   3.实验结果演示  这里我们开启了三个客户端一个服务端，服务端端口号为8888    每个client创建后输入自己的名字，会提示welcome XXX，当有新的client加入时，会提示xxx joinedt the chat。  用户可以发送消息，发送的消息会被显示在窗口中央。 | |
| 实验总结（遇到的问题及解决办法、体会）：  在这个实验中，实现了一个基于TCP的聊天小程序，在实现过程中使用了多线程技术，使得一个服务端能同时和多个客户端交流，客户端之间也能同时接受和发送消息。在实验中进一步了解了TPC协议在python中实现的流程，了解了即时通讯软件在实现过程中最基本的思想，比如群组消息如何实现。  在实现过程中使用了Tkinter这个库，这个库曾经接触不多，重新学习了下了解了GUI界面制作的基本流程，相较于控制台的程序，GUI更加直观使用起来也更加友好。  时间匆忙，代码还有很多可以改进的地方，比如可以用更加美观的GUI库来实现这个界面，比如Kivy等，比如实现Client之间P2P的交流等。我将会在后续空余时间改进代码。 | |
| 器材、工具领用及归还负责人：於文卓 | 实验记录人：於文卓 |
| 实验执笔人：於文卓 | 报告协助人：於文卓 |
| 小组成员签名：於文卓 | |
| 验收人： | 成绩评定： |