实验报告

|  |  |
| --- | --- |
| 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： |