计算机网络实验报告

姓名：白思雨

学号：2186123935

日期：2021 年 1 月 10 日

实验一 Socket网络编程实验

一、概述

1.1 实验目的

1）掌握Sockets的相关基础知识，学习Sockets编程的基本函数和数据类型
2）掌握UDP、TCP Client/Server模式的通信原理。
3）掌握socket编程命令

1.2 实验内容

1）实现一个简单的客户机/服务器程序,基于TCP和UDP协议分别实现。
2）应用场景为一个验证用户登录的程序。

1.3 实验要求

1）比较TCP DUP两种协议的不同，在实验报告中写出自己的理解；
2）可用多种语言实现，建议C/C++，JAVA或Python。

二、 实验过程及结果

2.1 实验步骤

本次实验采用Python编程语言编写
步骤1：编写server端程序
Server端分为TCP协议和UDP协议版本
TCP版本
监听本地8899端口，设置了运行中对应的提示语句，并对传入数据长度的有效性进行了判断

```python
import socket
sk = socket.socket()
sk.bind(('127.0.0.1', 8899))
sk.listen()
conn, addr = sk.accept()

true_name = 'abc'
true_passwd = '123'

tip1 = 'Name:'
tip2 = 'Passwd:'
tip3 = 'Authentication failed! Please input again!'
tip4 = 'Authentication is successful!'

```

```
while True:
    while True:
        conn.send(tip1.encode('utf-8'))
        name = conn.recv(1024)
        name = name.decode('utf-8')
        if len(name):
            break

    while True:
        conn.send(tip2.encode('utf-8'))
        password = conn.recv(1024)
        password = password.decode('utf-8')
        if len(password):
            break

    if name == true_name:
        if password == true_passwd:
            break
        else:
            conn.send(tip3.encode('utf-8'))
            continue
    else:
        conn.send(tip3.encode('utf-8'))
        continue

conn.send(tip4.encode('utf-8'))
print('Authentication is successful! Exit')
conn.close()
sk.close()
```

UDP版本

监听本地9090端口，先对客户端的合法性进行了一个简单鉴别，踢出不符合要求（不发送“hello”问好信息）的客户端。并设置了运行中对应的提示语句，对传入数据长度的有效性进行了判断

```
import socket
sk = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sk.bind(('127.0.0.1', 9090))

true_name = 'abc'
true_passwd = '123'

tip1 = 'Name:'
tip2 = 'Passwd:'
tip3 = 'Authentication failed! Please input again!'
tip4 = 'Authentication is successful!'

data, addr = sk.recvfrom(1024)
data = data.decode('utf-8')
```

```
16  while data != 'hello':
17      msg = 'error'
18      sk.sendto(msg.encode('utf-8'), addr)
19      sk.close()
20
21  while True:
22      sk.sendto(tip1.encode('utf-8'), addr)
23      name, addr = sk.recvfrom(1024)
24      name = name.decode('utf-8')
25      sk.sendto(tip2.encode('utf-8'), addr)
26      password, addr = sk.recvfrom(1024)
27      password = password.decode('utf-8')
28
29      if name == true_name:
30          if password == true_passwd:
31              break
32          else:
33              sk.sendto(tip3.encode('utf-8'), addr)
34              continue
35      else:
36          sk.sendto(tip3.encode('utf-8'), addr)
37          continue
38  sk.sendto(tip4.encode('utf-8'), addr)
39  print('Authentication is successful! Exit')
40  sk.close()
```

步骤2：编写client端程序

Client端也分为TCP协议和UDP协议版本

TCP版本

访问本地8899端口，同时在循环中加入对空字符、退出等特殊情况的判断

```
1   import socket
2   sk = socket.socket()
3   sk.connect(('127.0.0.1', 8899))
4
5   while True:
6       rst = sk.recv(1024)
7       rst=rst.decode('utf-8')
8       print(rst)
9
10      if rst == 'Authentication failed! Please input again!':
11          print('\n')
12          continue
13      if rst == 'Authentication is successful!':
14          break
15
16      msg = input()
17      if msg == 'quit':
18          print('Communication Ended')
19          break
20      while len(msg) == 0:
21          print('Error! Please input again!')
22          msg = input()
23      sk.send(msg.encode('utf-8'))
24
25  sk.close()
```

UDP版本

访问本地9090端口，并发送问好信息与服务端交互鉴权，同时在循环中加入对空字符、退出等特殊情况的判断

```python
import socket
sk = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

addr = ('127.0.0.1', 9090)
msg = 'hello'
sk.sendto(msg.encode('utf-8'), addr)
while True:
    rst, addr = sk.recvfrom(1024)
    rst = rst.decode('utf-8')
    print(rst)

    if rst == 'Authentication failed! Please input again!':
        print('\n')
        continue
    if rst == 'Authentication is successful!':
        break

    msg = input()
    if msg == 'quit':
        print('Communication Ended')
        break
    while len(msg) == 0:
        print('Error! Please input again!')
        msg = input()
    sk.sendto(msg.encode('utf-8'), addr)

sk.close()
```

步骤3：交互通信结果

TCP版本

TCP客户端在输入错误的名字、密码均返回错误提示，对于空也有较好的判断。输入正确则返回正确提示，并退出程序。

```
D:\code>python tcp_client.py
Name:
666
Passwd:
666
Authentication failed! Please input again!


Name:

Error! Please input again!
abc
Passwd:

Error! Please input again!
123
Authentication is successful!

D:\code>
```

TCP服务端在收到正确登录信息后，提示认证成功，同时也退出程序。

```
D:\code>python tcp_server.py
Authentication is successful! Exit
```

UDP版本

与TCP版本的相同，UDP客户端在输入错误的名字、密码均返回错误提示，对于空也有较好的判断。输入正确则返回正确提示，并退出程序。

```
D:\code>python udp_client.py
Name:
888
Passwd:
888
Authentication failed! Please input again!


Name:

Error! Please input again!
abc
Passwd:
123
Authentication is successful!
```

UDP服务端在收到正确登录信息后，提示认证成功，同时也退出程序。

```
D:\code>python udp_server.py
Authentication is successful! Exit
```

2.2 实验结果分析

1)服务器如何能实现循环监听？

在程序监听阶段使用while循环，保持持续监听。并根据客户端传入的信息进行判断、中断跳出循环。

2) 比较两种协议在代码层面的区别。

TCP协议提供了稳定可靠的连接。如下图代码所示，服务端在绑定端口后，listen函数把进程变为一个服务器，并指定相应的套接字变为被动连接。
TCP服务端的基本流程：socket->bind->listen->accept->send/recv->close

```
1  import socket
2  sk = socket.socket()
3  sk.bind(('127.0.0.1', 8899))
4  sk.listen()
5  conn, addr = sk.accept()
```

UDP协议的连接并不可靠，服务端直接绑定监听对应端口

```
1  import socket
2  sk = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
3  sk.bind(('127.0.0.1', 9090))
```

2.3 互动讨论主题

1) TCP协议和HTTP协议的区别和联系；

区别：TCP协议是传输层协议，主要解决数据如何在网络中传输；而HTTP协议是应用层协议，主要解决如何包装数据信息。

联系：TCP协议是HTTP协议的基础，HTTP协议利用TCP协议在两台主机(通常是服务器和客户端)之间传输信息。

2.4 进阶自设计

1) 在服务器端实现多线程的好处是什么？如何在服务端实现多线程？

好处：可以并发处理多个客户端的请求，支持双方自由发送和接收数据，提高信息传输处理的效率

实现：以本次使用的 Python 语言网络变成而言，可以引入 threading 库，即可实现服务端多线程

```
from threading import Thread
(省略部分代码内容)
While True:
    conn, addr = sk.accept()
    p = Thread(target =conn, recv(1024), args = (conn, addr))
    p.start()
```

实验二 RIP 路由协议软件实验

一、概述

1.1 实验目的

通过软件实现 RIP 协议，详细分析距离矢量算法，掌握网络协议的构建过程。

1.2 实验内容

在软件层面上构建网络拓扑，编写路由器等代码，然后基于网络拓扑实现 RIP 路由协议，一级相应分析结果，如何实现路由表动态更新等。

1.3 实验要求

实现路由协议的主要功能。

二、 实验过程及结果

2.1 实验过程

完成路由工具包 DVRouter 部分的代码，在其上实现 RIP 协议。利用邻接表，路由表完成对路由表项内容的建立，更新维护，通过路由表实现数据转发与信息交换的功能。

2.2 实验结果

查看主机h1 ping主机h2后的结果并查看路由表A的内容

```
>>> print(A.table)
=== Table for A ===
name   prt lat  sec
------ --- ---- -----
h1     0   1    inf
h2     1   3    13.80
h3     1   4    13.90
h4     1   5    14.11
>>> print(A.table)
=== Table for A ===
name   prt lat  sec
------ --- ---- -----
h1     0   1    inf
h2     1   3    13.29
h3     1   4    13.40
h4     1   5    13.60
>>> print(A.table)
=== Table for A ===
name   prt lat  sec
------ --- ---- -----
h1     0   1    inf
h2     1   3    12.69
h3     1   4    12.80
h4     1   5    13.00
```

```
>>> h1.ping(h2)
>>> DEBUG:user:h2:rx: <Ping h1->h2 ttl:17> A,B,h2
DEBUG:user:h1:rx: <Pong <Ping h1->h2 ttl:17>> B,A,h1
```

2.3 主要程序

```python
import sim.api as api
from cs168.dv import RoutePacket, \
    Table, TableEntry, \
    DVRouterBase, Ports, \
    FOREVER, INFINITY


class DVRouter(DVRouterBase):

    ROUTE_TTL = 15

    GARBAGE_TTL = 10

    SPLIT_HORIZON = False
    POISON_REVERSE = False
```

```
    POISON_EXPIRED = False

    SEND_ON_LINK_UP = False

    POISON_ON_LINK_DOWN = False

    def __init__(self):

        assert not (self.SPLIT_HORIZON and self.POISON_REVERSE), \
            "Split horizon and poison reverse can't both be on"

        self.start_timer()


        self.ports = Ports()


        self.table = Table()
        self.table.owner = self

        self.history = {}

    def add_static_route(self, host, port):

        assert port in self.ports.get_all_ports(), "Link should be up, but is not."


        tableEntry = TableEntry(
            host, port, self.ports.get_latency(port), FOREVER)
        self.table[host] = tableEntry

    def handle_data_packet(self, packet, in_port):


        dst = packet.dst
        if dst in self.table.keys():
            latency = self.table[dst][2]
            out_port = self.table[dst][1]
            if latency < INFINITY:
                self.send(packet, out_port)

    def send_routes(self, force=False, single_port=None):


        for host, tableEntry in self.table.items():
```

```
            port = tableEntry[1]
            if single_port is None:
                for out_port in self.ports.get_all_ports():
                    if self.SPLIT_HORIZON:
                        if out_port != port:
                            latency = tableEntry[2]
                            routePacket = RoutePacket(host, latency)
                            if not force:
                                if (out_port, host) not in self.history.keys() or s
elf.history[(out_port, host)].latency != routePacket.latency:
                                    self.send(routePacket, out_port)
                                    self.history[(out_port, host)
                                                 ] = routePacket
                            else:
                                self.send(routePacket, out_port)
                                self.history[(out_port, host)] = routePacket
                    elif self.POISON_REVERSE:
                        if out_port != port:
                            latency = tableEntry[2]
                        else:
                            latency = INFINITY
                        routePacket = RoutePacket(host, latency)
                        if not force:
                            if (out_port, host) not in self.history.keys() or self.
history[(out_port, host)].latency != routePacket.latency:
                                self.send(routePacket, out_port)
                                self.history[(out_port, host)] = routePacket
                        else:
                            self.send(routePacket, out_port)
                            self.history[(out_port, host)] = routePacket
                    else:
                        latency = tableEntry[2]
                        routePacket = RoutePacket(host, latency)
                        if not force:
                            if (out_port, host) not in self.history.keys() or self.
history[(out_port, host)].latency != routePacket.latency:
                                self.send(routePacket, out_port)
                                self.history[(out_port, host)] = routePacket
                        else:
                            self.send(routePacket, out_port)
                            self.history[(out_port, host)] = routePacket

            else:
                out_port = single_port
```

```
                if self.SPLIT_HORIZON:
                    if out_port != port:
                        latency = tableEntry[2]
                        routePacket = RoutePacket(host, latency)
                        if not force:
                            if (out_port, host) not in self.history.keys() or self.
history[(out_port, host)].latency != routePacket.latency:
                                self.send(routePacket, out_port)
                                self.history[(out_port, host)] = routePacket
                        else:
                            self.send(routePacket, out_port)
                            self.history[(out_port, host)] = routePacket
                elif self.POISON_REVERSE:
                    if out_port != port:
                        latency = tableEntry[2]
                    else:
                        latency = INFINITY
                    routePacket = RoutePacket(host, latency)
                    if not force:
                        if (out_port, host) not in self.history.keys() or self.hist
ory[(out_port, host)].latency != routePacket.latency:
                            self.send(routePacket, out_port)
                            self.history[(out_port, host)] = routePacket
                    else:
                        self.send(routePacket, out_port)
                        self.history[(out_port, host)] = routePacket
                else:
                    latency = tableEntry[2]
                    routePacket = RoutePacket(host, latency)
                    if not force:
                        if (out_port, host) not in self.history.keys() or self.hist
ory[(out_port, host)].latency != routePacket.latency:
                            self.send(routePacket, out_port)
                            self.history[(out_port, host)] = routePacket
                    else:
                        self.send(routePacket, out_port)
                        self.history[(out_port, host)] = routePacket

    def expire_routes(self):

        remove_hosts = []
        for host, tableEntry in self.table.items():
            expire_time = tableEntry[3]
            if(api.current_time() > expire_time):
```

```
                remove_hosts.append(host)
        if self.POISON_EXPIRED:
            for remove_host in remove_hosts:
                port = self.table[remove_host][1]
                self.table.pop(remove_host)
                tableEntry = TableEntry(remove_host, port, INFINITY, FOREVER)
                self.table[remove_host] = tableEntry
        else:
            for remove_host in remove_hosts:
                self.table.pop(remove_host)

    def handle_route_advertisement(self, route_dst, route_latency, port):

        is_poisoned = (route_latency == INFINITY)
        if is_poisoned:
            new_latency = INFINITY
        else:
            new_latency = route_latency + self.ports.get_latency(port)
        if route_dst not in self.table.keys():
            if not is_poisoned:
                self.table[route_dst] = TableEntry(
                    route_dst, port, new_latency, api.current_time() + self.ROUTE_T
TL)
                self.send_routes()
        else:
            current_latency = self.table[route_dst][2]
            current_port = self.table[route_dst][1]
            if new_latency < current_latency or port == current_port:
                if is_poisoned:
                    self.table[route_dst] = TableEntry(
                        route_dst, port, new_latency, api.current_time())
                    self.send_routes()
                else:
                    self.table[route_dst] = TableEntry(
                        route_dst, port, new_latency, api.current_time() + self.ROU
TE_TTL)
                    self.send_routes()

    def handle_link_up(self, port, latency):

        self.ports.add_port(port, latency)



        if self.SEND_ON_LINK_UP:
```

```
            self.send_routes(False, port)

    def handle_link_down(self, port):

        self.ports.remove_port(port)


        if(self.POISON_ON_LINK_DOWN):
            remove_hosts = []
            for host, tableEntry in self.table.items():
                if port == tableEntry[1]:
                    remove_hosts.append(host)
            for remove_host in remove_hosts:
                port = self.table[remove_host][1]
                self.table.pop(remove_host)
                tableEntry = TableEntry(remove_host, port, INFINITY, FOREVER)
                self.table[remove_host] = tableEntry

        self.send_routes()
```