中山大學 SUN YAT-SEN UNIVERSITY

计算机网络实验报告

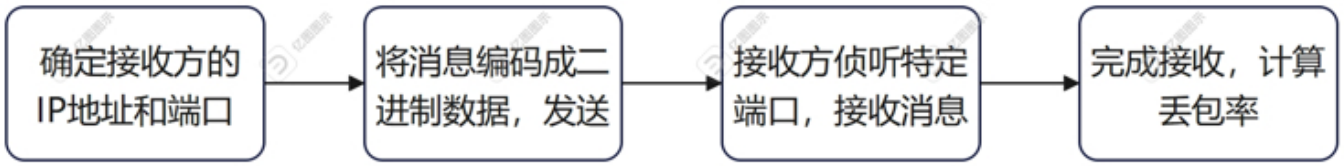
警示

1.实验报告如有雷同，雷同各方当次实验成绩均以 0 分计。
2.当次小组成员成绩只计学号、姓名登录在下表中的。
3.在规定时间内未上交实验报告的，不得以其他方式补交，当次成绩按 0 分计。
4.实验报告文件以 PDF 格式提交。

院系	计算机学院	班 级	计科（2）班	组长	郑梓霖
学号	21307077				
学生	凌国明				

UDP 编程实验

流程图



发送UDP数据包

```
import socket

def main():
    # 1.创建一个udp套接字
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # 2.准备接收方的地址
    count = 100
    while count > 0:
        udp_socket.sendto("hello".encode("utf-8"), ("172.16.17.2", 30000))
        count -= 1
    # 3.关闭套接字
    udp_socket.close()

if __name__ == "__main__":
    main()
```

1. socket.socket：套接字是一种网络编程中的通信工具，用于在计算机之间传输数据。创建一个UDP套接字（User Datagram Protocol）可以实现无连接的数据传输，适用于快速、简单的数据通信
2. udp_socket.sendto：字符串"hello"被编码为UTF-8编码格式的字节串。因为网络通信中需要传输字节数据而不是字符串。("172.16.17.2", 30000)是一个元组，其中包含了目标主机的IP地址和端口号，数据将被发送到IP地址为"172.16.17.2"的主机的30000端口上。

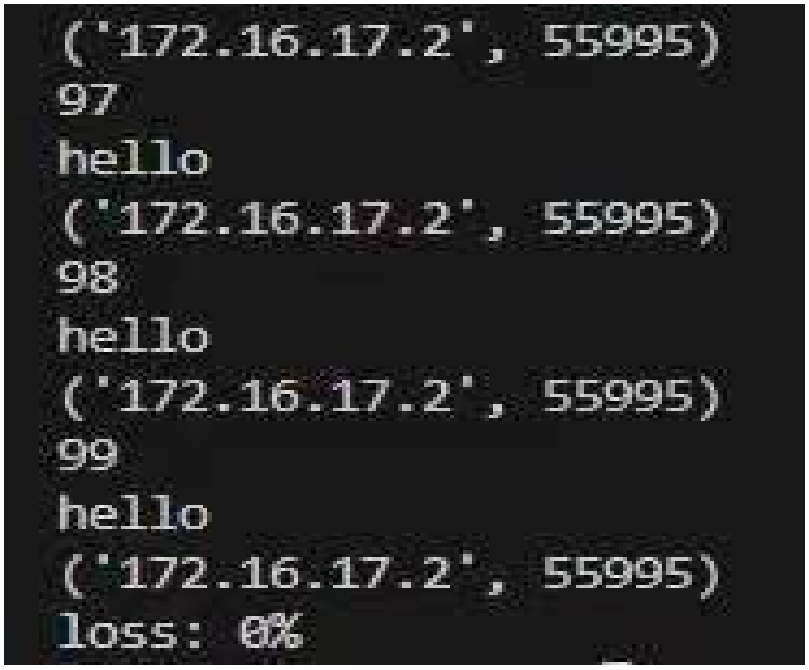
接收UDP数据包

```
import socket

def main():
    # 1.创建一个udp套接字
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # 2.绑定本地的相关信息
    local_addr = ("", 30000)
    udp_socket.bind(local_addr)
    counter = 0
    # 6. 显示对方发送的数据
    # 接收到的数据recv_data是一个元组
    # 第1个元素是对方发送的数据
    # 第2个元素是对方的ip和端口
    while counter < 100:
        # 3. 等待接收对方发送的数据
        recv_data = udp_socket.recvfrom(1024)
        # 1024表示本次接收的最大字节数
        print(counter)
        print(recv_data[0].decode('gbk'))
        print(recv_data[1])
        counter += 1
    print("loss: {}%".format(100-counter))
    # 3.关闭套接字
    udp_socket.close()

if __name__ == "__main__":
    main()
```

1. udp_socket.bind：将本地地址和端口与该UDP套接字绑定在一起。这意味着该套接字将监听本地计算机的IP地址（可以是任意可用地址）和端口号 30000，以便接收从该地址和端口发送过来的数据。

实验结果

python输出



WireShark抓包

udp and ip.dst == 172.16.17.3 and ip.src == 172.16.17.2

No.	Time	Source	Destination	Protocol	Length	Info
6361	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6362	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6363	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6364	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6365	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6366	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6367	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6368	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6369	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6370	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6371	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6372	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6373	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6374	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6375	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6376	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5
6377	49.270801	172.16.17.2	172.16.17.3	UDP	60	55995 → 30000 Len=5

问题与解决

网络太好，导致发送没有丢包

1. 尝试发送一万个数据包，仍然没有丢包
2. 尝试几台机子一起发送数据包，依然没有丢包

SOcket API 开发通信程序的函数

客户端程序

socket.socket(): 用于创建套接字对象，指定通信协议和类型（例如，TCP或UDP）。
socket.connect(): 用于连接到服务器，指定服务器的IP地址和端口号。
socket.send(): 用于向服务器发送数据。
socket.recv(): 用于从服务器接收数据。
socket.close(): 用于关闭套接字连接。

服务器程序

socket.socket(): 用于创建套接字对象，指定通信协议和类型（例如，TCP或UDP）。
socket.bind(): 用于绑定服务器的IP地址和端口号，以便客户端可以连接到它。
socket.listen(): 用于开始监听客户端连接请求。
socket.accept(): 用于接受客户端连接请求，并返回一个新的套接字对象，用于与客户端通信。
new_socket.send(): 用于向客户端发送数据。
new_socket.recv(): 用于从客户端接收数据。
new_socket.close(): 用于关闭与客户端的连接。

面向连接和面向非连接的客户端建立 Socket 的区别

面向连接

1. 使用socket.socket()创建套接字时，通常选择TCP协议。
2. 需要使用socket.connect()连接到服务器，指定服务器的IP地址和端口号。
3. 在连接建立后，可以使用socket.send()发送数据，并使用socket.recv()接收数据。

面向非连接

1. 使用socket.socket()创建套接字时，通常选择UDP协议。
2. 不需要使用socket.connect()建立连接，因为UDP是无连接的协议。
3. 直接使用socket.sendto()发送数据，需要指定目标服务器的IP地址和端口号。
4. 使用socket.recvfrom()接收数据，可以获取发送方的地址信息。

面向连接和面向非连接的客户端收发数据的区别

面向连接通常选择TCP协议，面向非连接通常选择UDP协议

面向连接

1. 数据发送前，需要建立连接使用socket.connect()，这意味着客户端和服务器之间有一个可靠的通信通道。
2. 使用socket.send()发送数据，该函数将确保数据按照发送顺序到达服务器，并且在必要时会进行重传以确保可靠性。
3. 使用socket.recv()接收数据，客户端会等待并接收来自服务器的数据，确保数据的完整性和有序性。
4. TCP协议提供了错误检测和重传机制，以保证数据的可靠传输

面向非连接

1. 不需要建立连接，直接使用socket.sendto()发送数据包，需要指定目标服务器的IP地址和端口号。
2. 使用socket.recvfrom()接收数据包，可以获取发送方的地址信息。
3. UDP不提供可靠性保证，发送的数据包可能丢失、重复或乱序到达，因此应用程序需要自行处理数据的完整性和顺序性。
4. UDP通常用于实时通信、广播和需要低延迟的应用，但不适用于要求可靠传输的场景。

面向非连接的客户端判断发送结束的方法

1. 固定长度数据包：在发送数据前，客户端和服务器都知道每个数据包的固定长度。当接收方收到足够长度的数据后，可以认为一个完整的数据包已经传输完毕。
2. 使用分隔符：在数据中添加特殊的分隔符，如换行符或特殊字符，以表示数据的结束。接收方通过检测分隔符来确定数据包的边界。
3. 使用消息头：在数据中包含消息头信息，消息头可以包含数据包的长度或其他元数据，以帮助接收方解析数据包。
4. 使用协议级别的结束标志：在应用层协议中定义特定的结束标志或消息，用于指示数据发送结束。接收方根据协议规定来解析数据包。

面向连接的通信 与 无连接通信 的对比

面向连接的通信

优点：

1. 可靠性：提供可靠的数据传输，确保数据的完整性、有序性和错误恢复，适合需要高度可靠性的应用。
2. 流量控制：具有流量控制机制，可以调整数据传输速率，防止网络拥塞。
3. 连接管理：建立和维护连接，支持点对点通信和双向通信，适合长时间通信的场景。

缺点：

1. 开销较大： 建立和维护连接需要额外的开销，可能会导致较高的延迟。
2. 不适用于实时性要求高的应用：由于连接建立和可靠性机制，不适用于要求低延迟和实时性的应用，如实时游戏或视频通话。

场合：

1. 文件传输、电子邮件、网页浏览、数据库连接等需要可靠性的应用场合。
2. 长时间的数据传输，如文件下载或上传。

无连接通信

优点：

1. 低延迟： 无连接通信通常具有较低的延迟，适合实时性要求高的应用，如实时游戏或视频流。
2. 简单： 无连接通信没有连接建立和维护的开销，通常较为简单。
3. 广播和多播： 支持广播和多播，可用于向多个接收者发送相同的数据。

缺点：

1. 不可靠，数据可能丢失、重复或乱序到达，需要应用程序自行处理。

场合：

1. 实时音视频通信、实时游戏、实时数据传输等需要低延迟和实时性的应用场合。
2. 广播或多播通信，例如视频流的分发。

Socket 工作方式的阻塞性

实验代码

```
udp_socket.sendto("hello".encode("utf-8"), ("172.16.17.2", 30000))
```

这一行代码是阻塞的。当使用UDP套接字发送数据时，套接字会尝试将数据发送到指定的目标地址和端口。如果网络条件良好，数据会很快发送出去，但如果网络延迟较大或出现其他问题，套接字可能会阻塞等待数据发送完成。

```
recv_data = udp_socket.recvfrom(1024)
```

这一行代码也是阻塞的。当使用UDP套接字接收数据时，套接字会等待并尝试接收来自其他主机的数据。如果没有数据到达，套接字将阻塞等待直到有数据到达或超过一定的超时时间。

阻塞与非阻塞的区别

阻塞：

1. 阻塞操作： 在阻塞模式下，套接字的操作（如发送和接收数据）会阻塞当前线程，直到操作完成或超时。这意味着如果没有数据可读取或无法发送数据，套接字将一直等待，阻止程序继续执行其他任务。
2. 等待数据： 当使用阻塞套接字接收数据时，套接字会一直等待，直到有数据到达或超过设置的超时时间。同样，在发送数据时，套接字会等待直到数据成功发送或发生错误。

非阻塞：

1. 非阻塞操作： 在非阻塞模式下，套接字的操作不会阻塞当前线程，而是立即返回，无论操作是否完成。如果操作不能立即完成，套接字会返回一个特定的错误或状态码，表示操作仍在进行中。
2. 轮询或事件驱动： 在非阻塞模式下，通常需要使用轮询或事件驱动的方式来检查套接字的状态，以确定何时可以继续进行操作。这可以通过循环检查套接字状态来实现，而不是等待。