实验 3-1 基于 UDP 服务设计可靠传输协议并编程实现

1813069 郭怡霏

一、实验目的：

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：

1. 建立连接；
2. 差错检测；
3. 确认重传。

流量控制采用停等机制，完成给定 txt 格式和 jpg 格式测试文件的单向传输（与 TCP 功能类似，在 UDP 服务上实现）。

二、实验原理：

UDP 是 User Datagram Protocol 的简称， 中文名是用户数据报协议，是 OSI（Open System Interconnection，开放式系统互联） 参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，IETF RFC 768 是 UDP 的正式规范。UDP 在 IP 报文的协议号是 17。

UDP 协议与 TCP 协议一样用于处理数据包，在 OSI 模型中，两者都位于传输层，处于 IP 协议的上一层。UDP 有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。UDP 用来支持那些需要在计算机之间传输数据的网络应用。包括网络视频会议系统在内的众多的客户/服务器模式的网络应用都需要使用 UDP 协议。UDP 协议从问世至今已经被使用了很多年，虽然其最初的光彩已经被一些类似协议所掩盖，但即使在今天 UDP 仍然不失为一项非常实用和可行的网络传输层协议。

许多应用只支持 UDP，如：多媒体数据流，不产生任何额外的数据，即使知道有破坏的包也不进行重发。当强调传输性能而不是传输的完整性时，如：音频和多媒体应用，UDP 是最好的选择。在数据传输时间很短，以至于此前的连接过程成为整个流量主体的情况下，UDP 也是一个好的选择。

总而言之，UDP 的特点是发送快速但不一定传输完整，且不支持差错检测、

确认重传等功能，因此将这些功能定义为上层的协议，通过携带校验和、不断地单向传输不同的数据包来实现。

三、实验重点：

1. 建立连接；
2. 差错检测与损坏包重传；
3. 应用层的自定义协议。

四、实验环境：

1. 操作系统：Oracle VM VirtualBox 6.1.14 r140239 (Qt 5.6.2)
   Ubuntu 18.04.5 LTS (64 bit)
2. IDE：Visual Studio Code 1.51.1
3. 使用语言：C++
4. 使用库：stdio.h string.h stdlib.h unistd.h sys/types.h sys/socket.h sys/time.h netinet/in.h time.h pthread.h arpa/inet.h iostream fstream cstdlib ctime

五、实验步骤：

自定义协议：

1. 建立连接：client 获取到服务器的地址之后向 server 发送想要的文件名；server 端创建 UDP 套接口、创建数据报 socket、绑定套接口之后开始监听，如接收到 client 的文件名请求后即生成新数据包并发送；
2. 传输数据类型：完整的数据包是一个名为 Pack 的类，包括包头 PackInfo 和正式报文 buf 两部分，包头中包括数据包序列号 id、报文长度 buf_size 和校验和 checksum，本次作业实现 server 向 client 传输文件，server 发送的是完整数据包，而 client 回发的 ack 只有包头，通过改变其中的 id 来告知所需数据包；
3. 读写方式：因为需要传输 txt 文本文档和 jpg 图片，因此一律使用二进制读写

形式；

4. 文件结束：server 设置为每次只能发送一个文件，因此如果创建新包时，使用 fread 读取数据得到返回值为负，则说明文件已经读完，则设置最后一个数据包的 id 为-1，发送给客户端之后不再接收 ack，直接关闭文件并退出；

5. 差错检测：server 在发送数据包之前通过计算来确定校验和 checksum，计算公式为 head.checksum = ~(SERVER_PORT & SERVER_IP_1 & SERVER_IP_2 & SERVER_IP_3 & SERVER_IP_4 & head.id & head.buf_size)（将 server 的端口号、IPv4 的四位、数据包 id、报文长度这七个数按位与之后取反）；client 不设置缓存，收到数据包之后检查计算得到的 checksum 是否正确，如计算结果不合则直接丢弃包，并返回该包的包头作为 ack；

6. 确认重传：server 端设置一个名为 data 的包来传输，使用停等机制，故收到的 ack 编号只会与将要发送的 id 相等或小 1。设置一个 while(1)的循环，每一次循环中先检查对方所需的数据包是否为将要发送的那一个（即有没有丢包错包情况）：如是，则将继续读取文件中长度不超过 BUFF_LEN（设为 1024）的数据作为新包报文，设置好包头之后发送给 client；否则，此时 data 的数据没有刷新，还是上一个包的数据，可更新校验和后重新发送（因为此时可能是错包的情况，checksum 计算有误，因此重新计算）；

7. 结束：client 收到 id 为-1 的数据包表明文件传输结束，不再接收并退出。

编译：

在 vscode 命令行打开当前文件夹，输入如下代码：

```
idaguo@idaguo-VirtualBox:~/codes/network/lab3-1$ g++ client.cpp -o client
idaguo@idaguo-VirtualBox:~/codes/network/lab3-1$ g++ server.cpp -o server
```

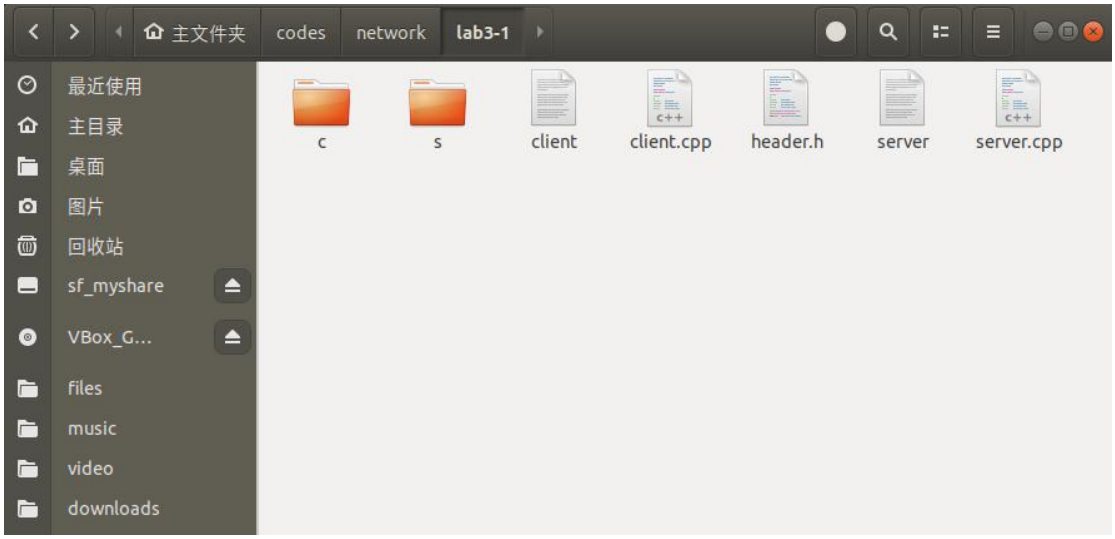
运行：

在命令行打开当前文件夹，输入如下代码，即可运行：

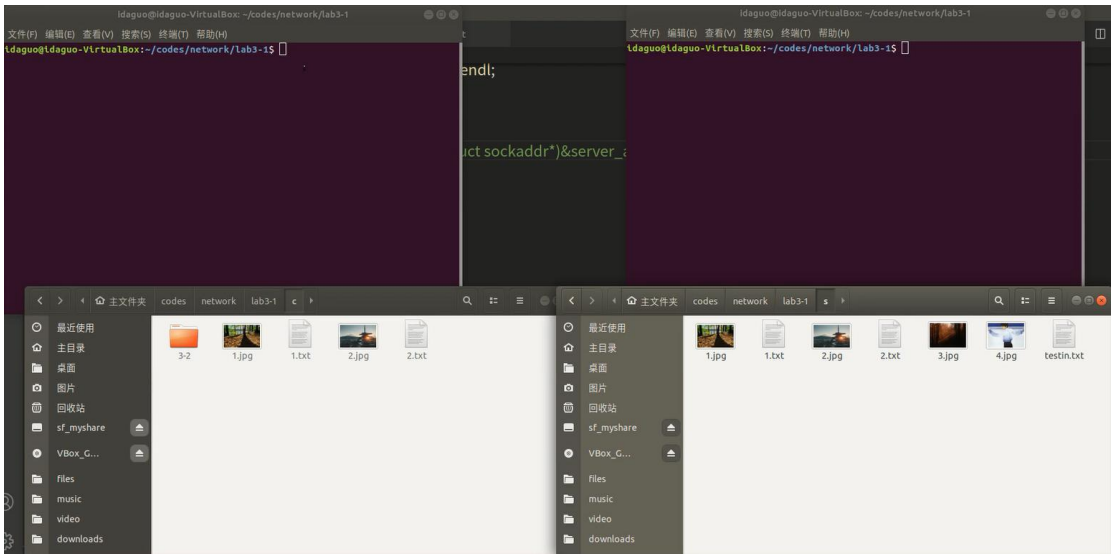
```
idaguo@idaguo-VirtualBox:~/codes/network/lab3-1$ ./server
```

```
idaguo@idaguo-VirtualBox:~/codes/network/lab3-1$ ./client
```

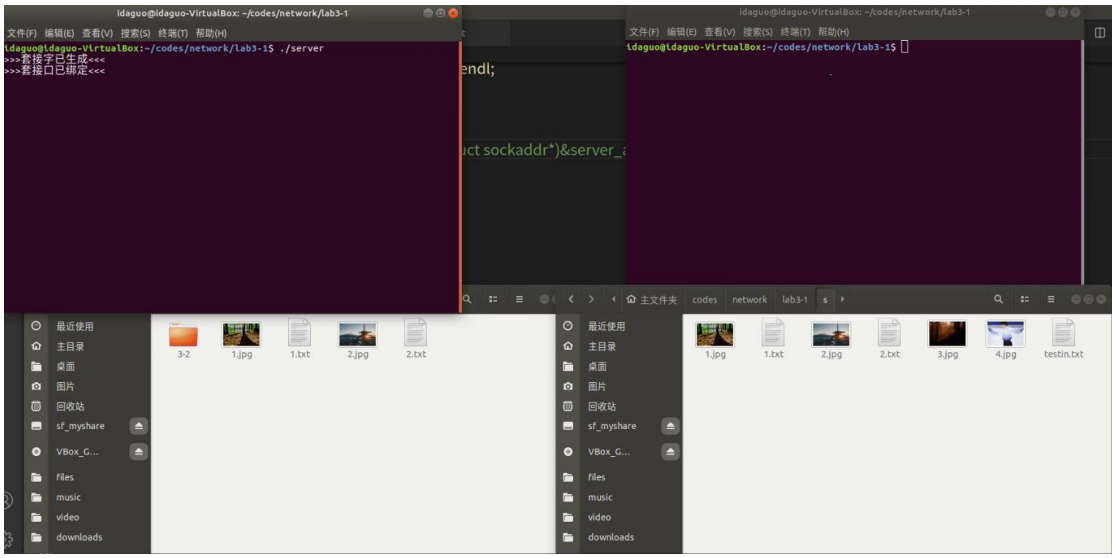
六、实验结果：



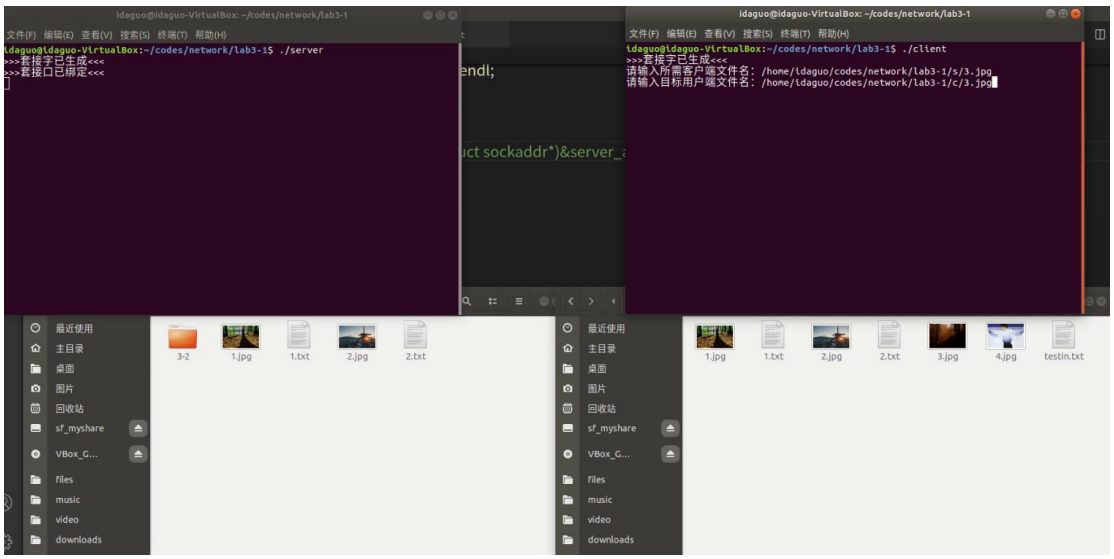
以上是文件列表，分别是客户端文件夹、服务器文件夹、客户端可执行文件、客户端源代码、服务器可执行文件、服务器源代码。通过在命令行输入./xxx（xxx为可执行文件名）来运行。



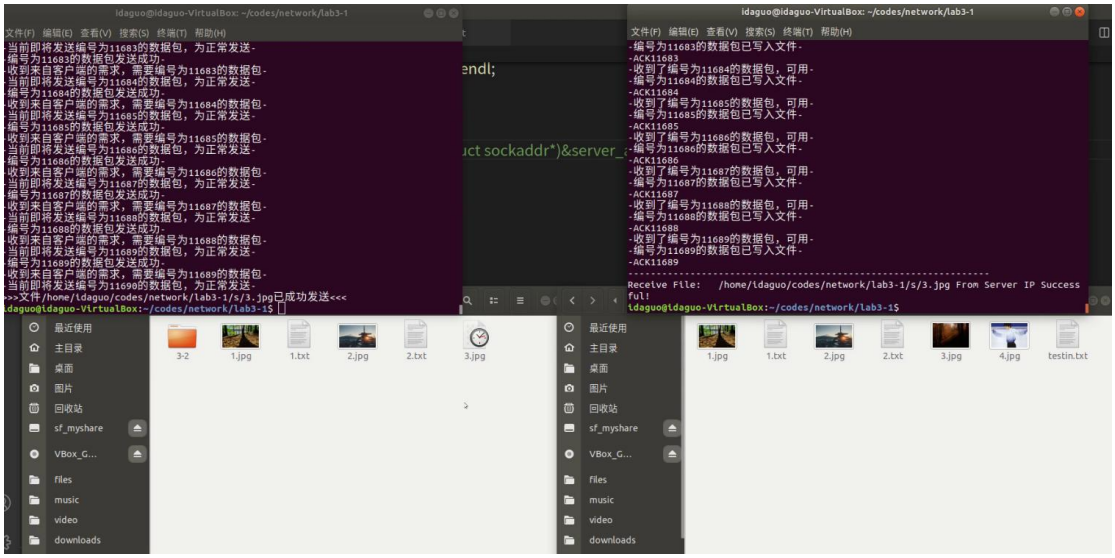
传输文件前的 client 文件夹和 server 文件夹



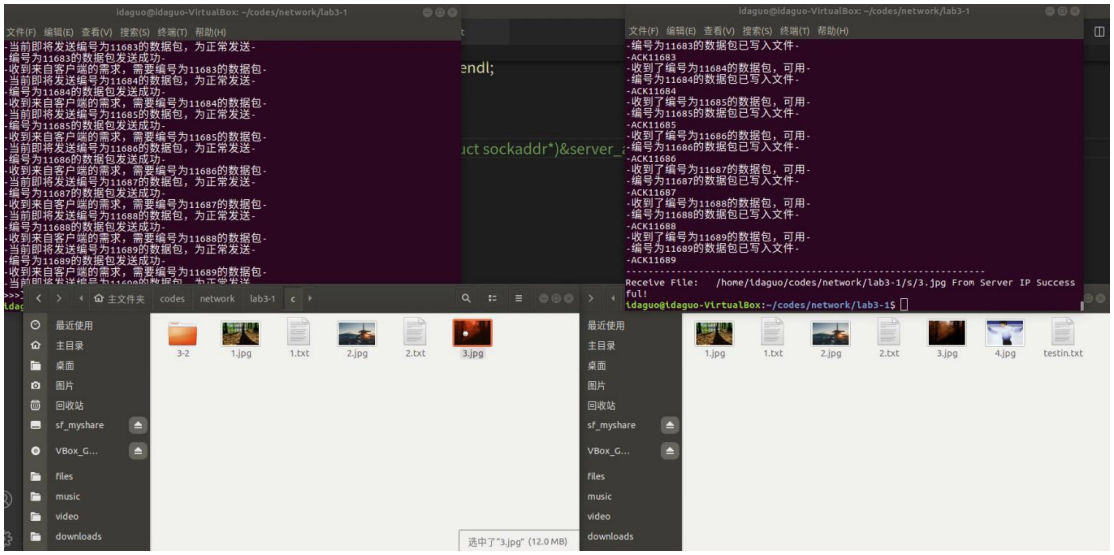
（1）服务器完成初始化设置开始监听



（2）客户端初始化，由用户输入希望得到的文件和要保存的位置



（3）名为 3.jpg 的文件传输结束，服务器和客户端都退出



（4）传输结束，client 文件夹有了被传过来的 3.jpg 文件，可正常显示

七、问题与不足：

起初设置 client 端只能接收一份文件，接收完自动退出，而 server 端不退出，但发现传输图片时会出现屏闪和 Ubuntu 的连续提示音，图片成功接收之后想再次运行 client，发现什么也传不了，将 server 重启后可以传 txt 文件但不能传图片，必须将 client 可执行程序删除后重新编译才可以，因此设置 server 也是自动退出。