实验 3-1 基于 UDP 设计可靠传输协议并编程实现

1813800　沈 哲

目录

1 实验要求 **2**

2 功能实现 **2**

2.1 协议设计 2
2.2 建立与断开连接 2
2.3 差错检测 2
2.4 确认重传和停等协议 3
2.5 其他设计 3

3 代码分析 (仅展示部分核心代码) **3**

3.1 公共头文件定义 3
3.2 发送端和接收端都使用的函数 4
3.2.1 makePackage 4
3.2.2 doCheckSum 5
3.2.3 CheckSEQ 5
3.2.4 IsACK,IsFIN,IsSYN 5
3.3 客户端（发送端） 5
3.4 服务端（接收端） 7

4 结果展示 **7**

4.1 发送端丢弃数据包 8
4.2 接收端丢弃 ACK 包 8
4.3 整体效果 9

5 总结反思 **9**

5.1 巩固了 C++ 基础知识 9
5.2 更熟悉了 UDP 和 TCP 协议以及基于 winsock 的编程 9

1 实验要求

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：建立连接、差错检测、确认重传。流量控制采用停等机制，完成给定测试文件的传输。

要求实现单向传输。对于每一个任务要求给出详细的协议设计。完成给定测试文件的传输，显示传输时间和平均吞吐率。

2 功能实现

2.1 协议设计

UDP 是传输层中面向无连接的协议，在编程上服务端和客户端是没有区别的，本实验实现从客户端（发送端）到服务端（接收端）的传输。

基于 UDP 设计可靠传输协议，可以参考 TCP 协议设计相应的数据报的字段，本实验设计的数据报分为两部分——头部和数据部分。数据部分占 65300 字节。头部包括：序列号、确序号、检验和字段、标志位字段、长度字段。每一部分占 2 字节，共占 10 字节。

```
Header:10 bytes
| 2 bytes  | 2 bytes  | 2 bytes  | 2 bytes  |  2 bytes  |
|   SEQ    |  ACKnum  | CheckSum |    ID    |   Length  |

ID:2 bytes
|  FIN  |  ACK  |  SYN  |
```

SEQ 是发送的序列号，主要在发送端使用，本次实验的停等协议中，只使用 0 和 1 两个序列号即可。

ACKnum 是确认序列号，主要在接收端使用，接收端回复的 ACKnum 等于发送端的 SEQ。

CheckSum 是检验和字段，发送端制作数据包时计算检验和填入，接收端也对收到的数据包进行差错检验。

ID 是多位标志位，包括 FIN（断开连接标志位）、ACK（确认标志位）、SYN（连接建立标志位）。

2.2 建立与断开连接

发送端发送只包含头部的包，SYN 位置 1，接收端接收到后发送 ACK，连接建立。同理通过发送 FIN 与 ACK 进行“挥手”断开连接。

2.3 差错检测

采用 UDP 校验和计算方法。发送端在发送数据包时进行 UDP 校验和计算，结果写入 CheckSum 位。接收端接收到数据包后也进行校验和计算，结果为 0xFFFF 则无错误。

2.4 确认重传和停等协议

发送端收到确认序号 ACKnum 后再次发送下一个数据包。如果发送端发送的数据包丢失，接收端一直等待接收（程序里是在 while 循环中不断执行 recvfrom 函数），不发送数据包，或者接收端发送的数据包丢失，都会导致发送端收不到 ACKnum，这时发送端就重传数据包。

具体来说，发送端发送的数据包序列号只有 0、1 两种，如果发送的数据包 SEQ=0, 则期望收到 ACKnum=0, 之后再发送 SEQ=1 的数据包。若收不到 ACKnum=0 的数据包，就重传 SEQ=0 的数据包。

对于接收端，收到某个序列号的数据包，发送确认序号，确认序号 ACKnum 等于序列号 SEQ。如果收到重复的数据包（根据接收到包的序列号判断），就丢弃，但要发送 ACKnum。

这就实现了确认重传，能够处理发送端和接收端的丢包问题。

2.5 其他设计

采用非阻塞模式，设置一个超时时间（设置为 10ms），超过此时间无响应就返回一个值（-1），这样编程更方便。经过测试，超时未响应 recvfrom 函数的返回值为-1，错误类型 WSAGetLastError() 的返回值为 10060。

```
// 设置发送超时
setsockopt(socket_client,SOL_SOCKET,SO_SNDTIMEO, (char *)&nNetTimeout,sizeof(
    int));
//设置接收超时
setsockopt(socket_client,SOL_SOCKET,SO_RCVTIMEO, (char *)&nNetTimeout,sizeof(
    int));
```

利用随机数设置丢包，让发送端不发送数据包或者接收端不发送 ACK 数据包。测试传输的可靠性。

3 代码分析 (仅展示部分核心代码)

3.1 公共头文件定义

```
//设置IP和端口号等
#define server_Port 1001
#define server_IP "127.0.0.1"
#define client_Port 1002
#define client_IP "127.0.0.1"
//rand()%RAND_MOD_NUM 对随机数取模来决定发送还是丢弃数据包
int RAND_MOD_NUM=15;
//非阻塞模式的超时时间
int nNetTimeout=10;//毫秒
//缓冲区大小
```

```
#define BUF_LEN 65310//比2**16小一点
#define HEADER_LEN 10//头部长度
#define DATA_LEN 65300
/*
头部的设计
Header:10 bytes
| 2 bytes  | 2 bytes  | 2 bytes  | 2 bytes  |  2 bytes  |
|   SEQ    |  ACKnum  | CheckSum |    ID  |   Length  |

ID:2 bytes
|   FIN    |  ACK  |  SYN  |
*/
struct HEADER
{
  short SEQ;//序列号
  short ACKnum;//确认序号
  short CheckSum;//检验和
  short ID;//多个标志位
  short Length;//数据长度，这是基本固定的
};
//各个标志位
#define SYN 0x1//建立连接
#define ACK 0x2//确认
#define FIN 0x4//断开连接
#define ID_bit 0x6
#define ACKnum_bit 0x2
#define SEQ_bit 0x0
#define Length_bit 0x8
```

3.2 发送端和接收端都使用的函数

3.2.1 makePackage

用于制作数据包的头部，发送端需要设置校验和、发送序列号、数据长度。接收端需要设置确认序号。

```
header.Length = length;
header.SEQ =(1+header.SEQ)%2;//0 1 0 1
header.CheckSum = 0;
memcpy(&sendbuf, &header, HEADER_LEN);
```

注意建立连接时填充 SYN 字段，结束连接时填充 FIN 字段。发送端在读取数据后，需要再次填充 CheckSum 字段。

3.2.2 doCheckSum

计算 UDP 校验和的函数。发送端计算校验和填入 CheckSum 字段。接收端计算校验和判断是否等于 0xFFFF，不等于 0xFFFF 则数据出错。

```
//先将char型发送缓冲区转换成为16位的unsigned short型数据
//需要分情况，考虑缓冲区长度
array = new unsigned short[count];
for (int i = 0; i < count; i++)
{
    memcpy(&array[i], &sendbuf[i * 2], 2);
}
//计算校验和
while (count--)
{
    sum += array[i++];
    sum = (sum >> 16) + (sum & 0xFFFF);
}
header.CheckSum=~sum;
memcpy(&sendbuf[0],&header, 10);
```

3.2.3 CheckSEQ

发送端检查接收的 ACKnum 是否等于发送序列号。接收端检查接受的是否是新的序列号的数据包（若不是需要丢弃）。

```
memcpy(&seq, &recvbuf[ACKnum_bit], 2);//ACKnum字段
if (seq == header.SEQ)
    return true;
```

3.2.4 IsACK,IsFIN,IsSYN

检查这些标志位是否置位。

3.3 客户端（发送端）

先是 WinSocket 的初始化，创建 socket，设置超时时间等操作。

```
// 设置发送超时
setsockopt(socket_client,SOL_SOCKET,SO_SNDTIMEO, (char *)&nNetTimeout,sizeof(
    int));
//设置接收超时
setsockopt(socket_client,SOL_SOCKET,SO_RCVTIMEO, (char *)&nNetTimeout,sizeof(
    int));
```

发送包含 SYN 的包，准备建立连接。

连接建立后，先发送文件名。之后开始读文件（每次读取数据缓冲区大小的数据量）。制作数据包（包括头部的各个字段，SEQ 进行加 1 模 2 操作实现 01 交替，检待装入数据后计算校验和）。发送数据包，这里随机丢弃进行测试。

```
file.read(&sendbuf[10], DATA_LEN);
makePackage(file.gcount());
doCheckSum(file.gcount());
//发送数据包,随机丢弃（不发送）
if(rand()%RAND_MOD_NUM!=0)//遇到rand()%15==0丢弃
    sendto(socket_client, sendbuf, BUF_LEN, 0, (SOCKADDR*)&server_addr, sizeof
        (SOCKADDR));
else
    cout<<"丢弃数据包！！！！！！！！！！"<<endl;
```

接收到 ACK，并且确认序列号正确。

```
cout << "第" << trans_num << "次传输成功！" <<endl<< "传输" << file.gcount()
    <<"字节" << endl;
trans_bytes+=file.gcount();
memset(&sendbuf[10], 0, DATA_LEN);//清空sendbuf
continue;
```

未接收到 ACK，进行超时重传。

```
//接收ACK
recv_Ret = recvfrom(socket_client, recvbuf, HEADER_LEN, 0, (SOCKADDR*)&server_
    addr, &recv_para_len);
if (recv_Ret < 0)
{ //(WSAGetLastError() == 10060)
    cout << "超时，未接收到ACK，重传数据包！！！" << endl;
    /*处理:
    1.读文件指针回退,
    2.trans_num--
    3.接收端做丢弃acknum，就要处理来自发送端的重复的包，就必须检查序列号，发送
        端就要做好0 1交替
    调用两次makePackage即可！！！
    刚刚发了包0，多调用一次makePackage（SEQ==1）,再发包0时就可以了makePackage
        （SEQ==0）
    */
    file.seekg(-file.gcount(),ios::cur);//向前移动
    trans_num--;
    makePackage(0);//重传的处理，多调用一次使得序列号和之前一样
    continue;
}
```

3.4 服务端（接收端）

开始处理和发送端相同。建立连接后接收文件名。之后开始接收数据包，写入文件，发送 ACK 确认数据包（也进行随机丢弃）。

CheckSum 函数进行校验和计算，CheckSEQ 确保接收到的是下一个序列的数据包，否则接收重复丢弃即可，IsFIN 判断是否传输完成。

接收数据包并写入文件。

```
file.write(&recvbuf[10], DATA_LEN);
memcpy(&data_len, &recvbuf[Length_bit], 2);
trans_bytes += data_len;
memset(&recvbuf[10], 0, DATA_LEN); //及时清空缓存区
makePackage();
if(rand()%RAND_MOD_NUM!=0)
{
    sendto(socket_server, sendbuf, HEADER_LEN, 0, (SOCKADDR *)&client_addr,
        sizeof(SOCKADDR));
}
else
{
    cout<<"丢弃ACK包！！！！！！！！！！！"<<endl;
}
continue;
```

接收到重复的数据包，不写入文件，丢弃即可。

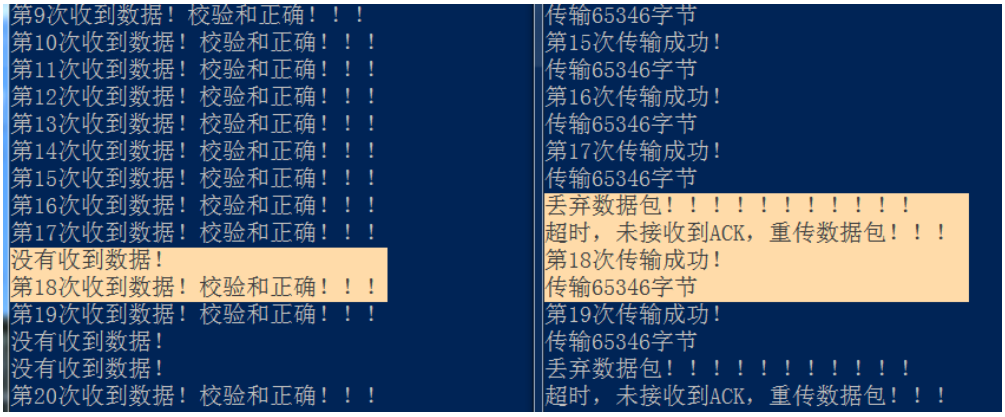
```
//本该收包1（然后回复acknum==1），结果收到包0，
//由于刚刚收到的包0，此时header.acknum==0，正常调用makePackage会使得header.
    acknum==1
//但是这时候还要回复acknum==0，函数细节-->memcpy(&header.ACKnum, &recvbuf[SEQ_
    bit], 2);
//还是直接用makePackage即可，因为是根据收到的recvbuf[SEQ_bit]确定acknum，
//不像发送端的SEQ是每次调用makePackage会01交替
cout<<"接收到重传的数据包，丢弃，发送ACK！！！"<<endl;
makePackage();
sendto(socket_server, sendbuf, HEADER_LEN, 0, (SOCKADDR *)&client_addr, sizeof
    (SOCKADDR));
memset(&recvbuf[10], 0, DATA_LEN); //及时清空缓存区
continue;
```

4 结果展示

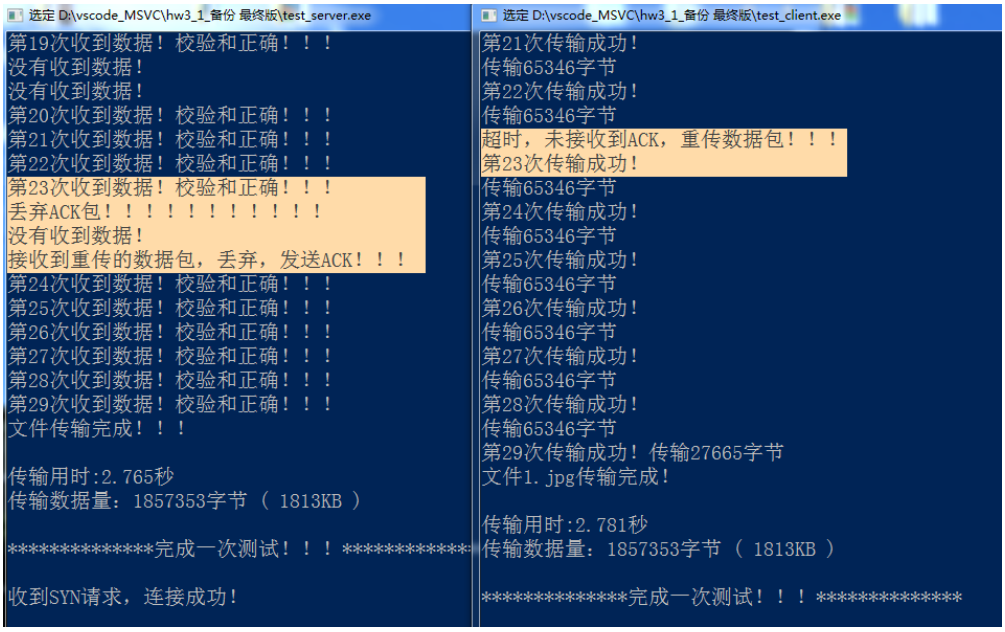
程序可以连续发送任意类型的文件，之后也可以设置用户输入 IP 地址、端口号，文件名，选择设置丢包率（随机数设置）、超时时间等等，能够实现文件的无损正确传输。超时时间越短（这里是

10ms），缓冲区越大（这里是比 2 的 16 次方小一些，不能超过，取 65310），文件发送速度越快。

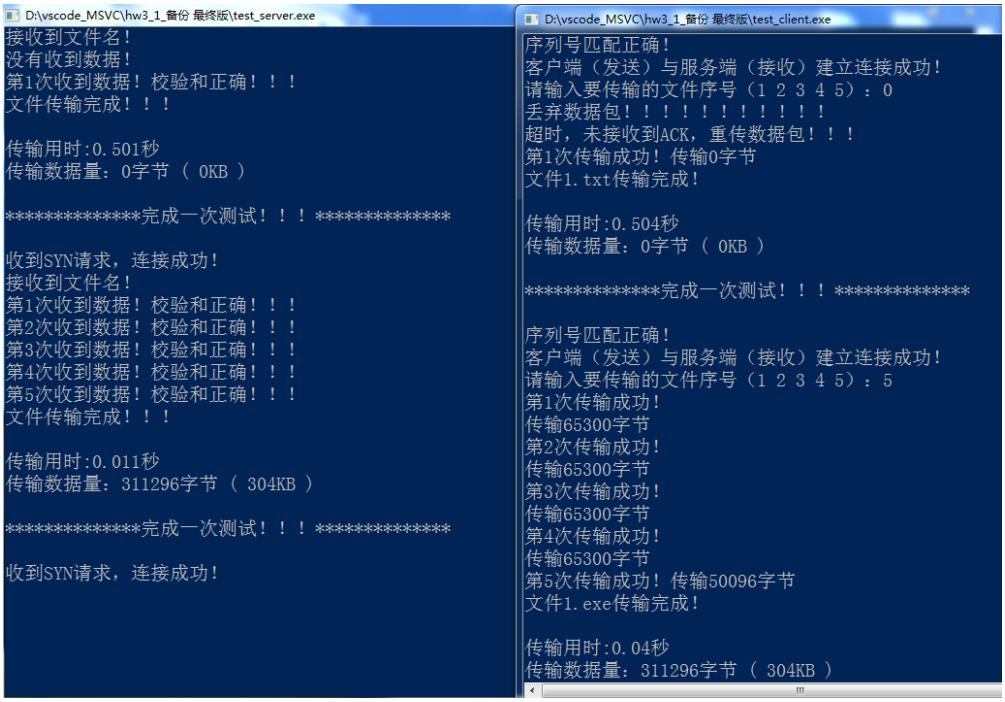
4.1 发送端丢弃数据包



4.2 接收端丢弃 ACK 包



4.3 整体效果



5 总结反思

5.1 巩固了 C++ 基础知识

在实验过程中因代码不严谨出现许多错误，通过调试更加熟悉了字符串函数的使用（如 memcpy），文件读写操作等等。

5.2 更熟悉了 UDP 和 TCP 协议以及基于 winsock 的编程

UDP 是传输层中面向无连接的协议，本实验以 UDP 为平台，进行可靠传输协议的设计，还是能够参考 TCP 的一些思想，对两种协议的了解也更深入。在设计思考过程中，复习了 rdt 不同版本的协议，对可靠传输协议的设计更加熟悉。