南開大學
Nankai University

基于 UDP 服务可靠传输协议编程实现（3-1）

姓 名：　边　笛

学 号：　2012668

学 院：计算机学院

目 录

一、实验内容说明

本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、确认重传等。流量控制采用停等机制，完成给定测试文件的传输。

二、实验设计

1. 数据报套接字

实验使用用户数据报协议 UDP 作为传输协议，它是一种无连接、不可靠的传输协议。为实现传输协议，在初始化套接字是将协议设置为 **IPPROTO_UDP**，数据传输使用 sendto、recvfrom 实现无连接数据传输。

UDP 是面向报文的，在设计时对于传输的数据报做了如下设计：

Scr_Port 源端口号	Dst_Port 目的端口号	
seq 序列号	ack 确认号	
datalen 数据长度	Flags 标志位	CheckSum 校验和
Data 数据		

同时服务器与客户端产生用于进行数据校验的伪首部，与数据报一起用于校验和的计算，对于数据的正确性作检查。

Scr_IP 源地址		
Dst_IP 目的地址		
Zero 零位	Protocol 协议	Len 长度

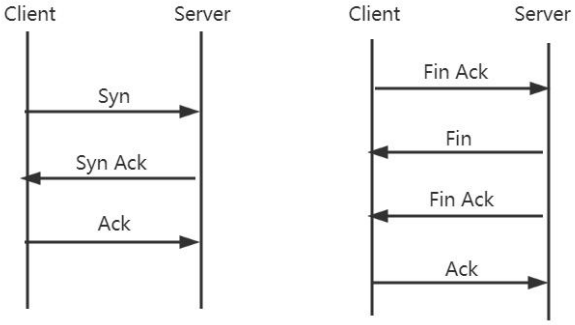
```
struct Message
{
    unsigned short src_port;
    unsigned short dst_port;
    unsigned int seq;
    unsigned int ack;
    unsigned int datalen =0;
    unsigned short Flags = 0;
    unsigned short CheckSum;
    char Data[8192]{};
```

```
struct Header
{
    unsigned long src_IP;
    unsigned long dst_IP;
    char zero = 0;
    int Protocol = 17;
    int length = sizeof(struct  Message);
};
```

2. 建立连接

建立连接与断开连接过程参考了 TCP 的三次握手四次挥手进行设计。

如果想要建立连接，需要完成如下消息传递：



3. 差错检验

为保证数据的可靠性，需要进行差错检验。在这里使用纠错码进行差错检验。使用到前面说明的数据报与伪首部来计算校验和。

在数据传输之前，需要借助 Message 结构体的 void setChecksum(struct Header* h) 进行校验和的设置，基本思路如下：

1. 产生伪首部，将设置校验和域清零。
2. 将伪首部与数据包一起看成 16 位整数序列，进行 16 位二进制反码求和。
3. 将其算结果取反写入校验和域段。

```
void Message::setChecksum(struct Header* h)
{
    this->CheckSum = 0;

    unsigned long sum = 0;
    int i=0;
    int count = sizeof(struct Header) / 2;
    while(count--)
    {
        sum += ((unsigned short*)h)[i];
        i++;
        if (sum & 0xffff0000)
        {
            sum &= 0x0000ffff;
            sum++;
        }
    }
    i=0;
    count = sizeof(struct Message) / 2;
    while(count--)
    {
        sum += ((unsigned short*)this)[i];
        i++;
        if (sum & 0xffff0000)
        {
            sum &= 0x0000ffff;
            sum++;
        }
    }
    this->CheckSum = ~(sum & 0xffff);
}
```

在接收到数据报后，接收端需要对数据报进行检查，对整个数据报反码求和。根据校验和的设置，可以判断当相加计算结果位全部为 1 说明没有检测到错误。基本思路如下：

1. 产生伪首部。
2. 按 16 位整数序列反码求和。
3. 判断计算结果，如果全 1 说明没有检测到差错，返回 true。

```
bool Message::Check(struct Header* h)
{
    unsigned long sum = 0;
    int i=0;
    int count = sizeof(struct Header) / 2;
    while(count--)
    {
        sum += ((unsigned short*)h)[i];
        i++;
        if (sum & 0xffff0000)
        {
            sum &= 0x0000ffff;
            sum++;
        }
    }
    i=0;
    count = sizeof(struct Message) / 2;
    while(count--)
    {
        sum += ((unsigned short*)this)[i];
        i++;
        if (sum & 0xffff0000)
        {
            sum &= 0x0000ffff;
            sum++;
        }
    }
    return sum == 0x0000ffff;
}
```

同时在数据传递时会进行序列号 Seq 与确认号Ack 的判断。Ack 是上一次接收的数据报的序列号加一，Seq 是上一次发送的数据报序列号加一。在接收和发送数据时，对 Seq 和 Ack 进行判断，如果 Seq 与 Ack 错误不做接收，等待重传。

4. 确认重传

确认重传采用 rdt3.0 的超时重传方式。在程序中设置了一个最大等待时间，在消息发出后会开始计时，如果超过最大等待时间就会重新发送上一条发送的数据报。

对于服务器和客户端发送的每条数据报都有设重传机制，这样既可以避免差错，也可以避免丢失。为了保证重传正常进行，专门设有一个 Message 用来储存上一条发送的数据报。

对于文件内容传输时的重传进行展示：

```
while(recvfrom(Server, (char*)&recvcontent, sizeof(struct Message), 0, (sockaddr*)&Client_addr, &l) == SOCKET_ERROR)
{
    if(clock() - filetimer >= Max_waitTime)
    {
        printTime();
        std::cout<<"[ Log ] Timeout! Retransmit data."<<endl;
        //重传
        while(1)
        {
            if (lastsend->getName())
            {
                while(1)
                {
                    int s = sendto(Server, (char*)lastsend, sizeof(struct Message), 0, (sockaddr*)&Client_addr, l);
                    if (s == SOCKET_ERROR)
                    {
                        printTime();
                        std::cout<<"[ Error ] Filed To Send File Name Ack"<<endl;
                        continue;
                    }
                    else
                    {
                        printTime();
                        std::cout<<"[ Log ] Send File Name Ack"<<endl;
                        printMess(*lastsend,SEND);
                        lastsendSEQ = lastsend->seq;
                        filetimer = clock();  //不退出，计时
                        break;
                    }
                }
            }
```

```
        else
        {
            while(1)
            {
                int s = sendto(Server, (char*)lastsend, sizeof(struct Message), 0, (sockaddr*)&Client_addr, 1);
                if (s == SOCKET_ERROR)
                {
                    printTime();
                    std::cout<<"[ Error ] Filed To Send File Name Ack"<<endl;
                    continue;
                }
                else
                {
                    printTime();
                    std::cout<<"[ Log ] Send File Data Block "<<packnum<<" Ack"<<endl;
                    printMess(*lastsend,SEND);
                    lastsendSEQ = lastsend->seq;
                    filetimer = clock();
                    break;
                }
            }
        }
    }
```

lastsend 存储了上一条发送的数据报。

5. 数据报传输

在进行传输时，数据都存储在 Message 的 Data 里。但由于文件大小不定，数据很可能无法由一个数据报完成传输。对此需要对文件进行划分传输。

首先计算所需数据报的个数：使用文件总长 除以 Data 的最大长度 向上取整。

由于可能不能整除，最后一个数据报的数据段就有可能是不满的，需要补 0 补全。

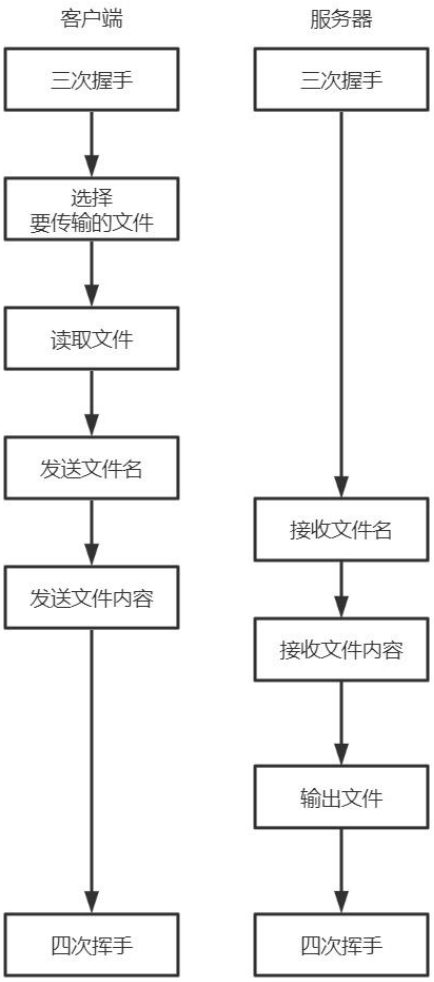
此外，最后一个数据报表示文件传输结束，需要单独设置 Psh 标志位，用以告诉接收端数据传输完毕可以进行文件数据的读取与输出了。

对此在发送端需要对最后一个数据报进行单独置位；在接收端接收数据报时要注意判断 Psh 标志位，即使对完成传输的文件进行读取。

由于本实验完成的是文件传输，在发送文件内容之前，首先发送了一条 Data 存储着传输文件文件名的数据报作为数据头，标示了传输文件的信息。

三、程序流程说明

客户端与服务器的运行流程如下所示：

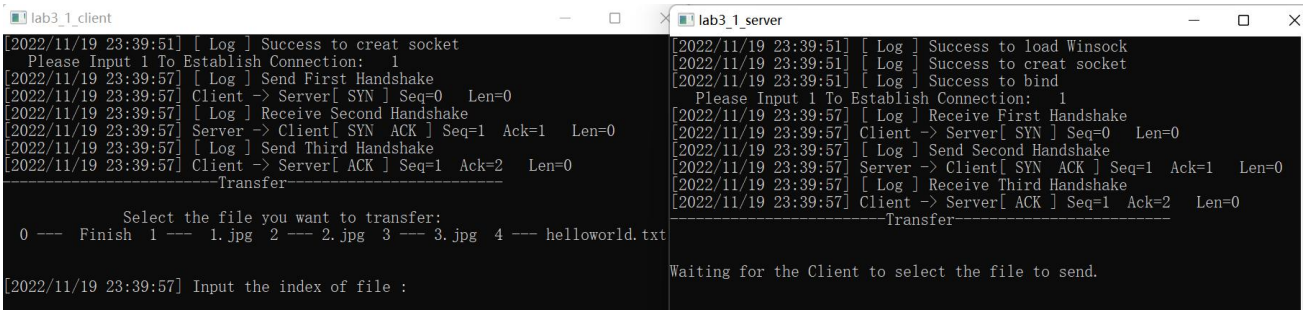


根据所设计的停等协议，客户端在发送数据后，会等待服务器接收数据报并发送回 Ack，之后才会继续发送下一跳数据。

四、运行结果展示

对于测试样例中的四条个文件都进行了传输：

三次握手



传输 1.jpg

```
[2022/11/19 23:36:50] Server -> Client[ ACK ] Seq=228  Ack=229   Len=0
[2022/11/19 23:36:50] [ Log ] Send File Data Block 227
[2022/11/19 23:36:50] Client -> Server[ PSH  ACK ] Seq=229  Ack=229   Len=5961
[2022/11/19 23:36:50] [ Log ] Receive File Data Block 227 Ack
[2022/11/19 23:36:50] Server -> Client[ ACK ] Seq=229  Ack=230   Len=0
```

传输 2.jpg

```
[2022/11/19 23:37:18] [ Log ] Send File Data Block 721
[2022/11/19 23:37:18] Client -> Server[ PSH  ACK ] Seq=951  Ack=951   Len=265
[2022/11/19 23:37:18] [ Log ] Receive File Data Block 721 Ack
[2022/11/19 23:37:18] Server -> Client[ ACK ] Seq=951  Ack=952   Len=0
```

传输 3.jpg

```
[2022/11/19 23:37:47] [ Log ] Send File Data Block 1462
[2022/11/19 23:37:47] Client -> Server[ PSH  ACK ] Seq=2414  Ack=2414   Len=482
[2022/11/19 23:37:47] [ Log ] Receive File Data Block 1462 Ack
[2022/11/19 23:37:47] Server -> Client[ ACK ] Seq=2414  Ack=2415   Len=0

[2022/11/19 23:37:47] Input the index of file :
```

传输 helloworld.txt

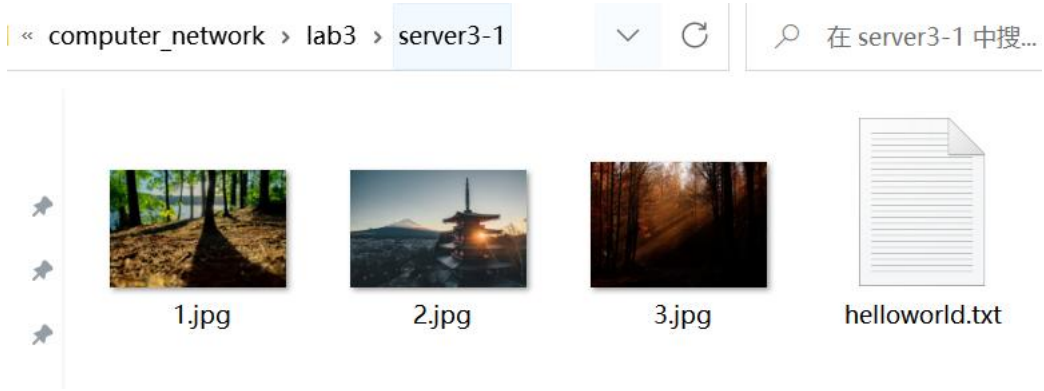
```
[2022/11/19 23:38:10] Client -> Server[ PSH  ACK ] Seq=2605  Ack=2605   Len=4032
[2022/11/19 23:38:10] [ Log ] Receive File Data Block 190 Ack
[2022/11/19 23:38:10] Server -> Client[ ACK ] Seq=2605  Ack=2606   Len=0

[2022/11/19 23:38:10] Input the index of file :
```

四次挥手

```
[2022/11/19 23:38:44] Receive First Handwave
[2022/11/19 23:38:44] Client -> Server[ FIN  ACK ] Seq=2607  Ack=2607   Len=0
[2022/11/19 23:38:44] [ Log ] Send Second Handwave
[2022/11/19 23:38:44] Server -> Client[ ACK ] Seq=2607  Ack=2608   Len=0
[2022/11/19 23:38:44] [ Log ] Send Third Handwave
[2022/11/19 23:38:44] Server -> Client[ FIN  ACK ] Seq=2608  Ack=2608   Len=0
[2022/11/19 23:38:44] [ Log ] Receive Fourth Handwave
[2022/11/19 23:38:44] Client -> Server[ ACK ] Seq=2608  Ack=2609   Len=0
请按任意键继续. . .
```

```
[2022/11/19 23:38:44] [ Log ] Send First Handwave
[2022/11/19 23:38:44] Client -> Server[ FIN  ACK ] Seq=2607  Ack=2607   Len=0
[2022/11/19 23:38:44] [ Log ] Receive Second Handwave
[2022/11/19 23:38:44] Server -> Client[ ACK ] Seq=2607  Ack=2608   Len=0
[2022/11/19 23:38:44] Receive Third Handwave
[2022/11/19 23:38:44] Server -> Client[ FIN  ACK ] Seq=2608  Ack=2608   Len=0
[2022/11/19 23:38:44] Send Fourth Handwave
[2022/11/19 23:38:44] Client -> Server[ ACK ] Seq=2608  Ack=2609   Len=0
请按任意键继续. . .
```

目标文件夹



五、问题与分析

在本次实验的代码编写过程中主要遇到的问题有以下两点。

①缓冲区设置：在文件传输过程中有很多需要用到字符数组、字符指针的地方。在程序编写的时

候需要对于这些区域特别注意，以免出现数组区域被污染的情况，需要时刻记得 memset。实际代码编写过程中的很多 bug 都出自于数组大小设置与区域置零。

②重传逻辑设置。在传输过程中需要时刻考虑出错重传的问题，尽可能地将数据报收发写在循环里。

通过本次实验完成了一次简单的传输协议设置，在代码编写过程中发现了许多对于课堂知识掌握不牢固的地方，并且在理论落地上存在着许多困难，还需要培养好自己的实践能力，能够将理论知识应用于实践。