

计算机网络第三次实验报告

Lab3-1 基于 UDP 服务设计可靠传输协议

2212422 孙启森

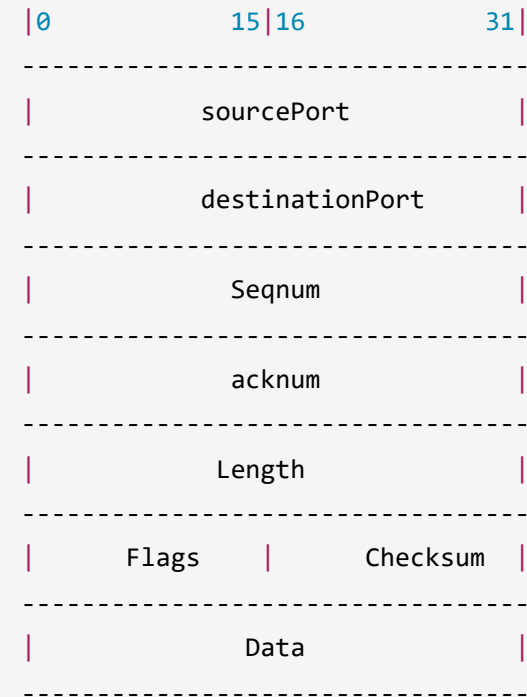
一、实验内容

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、接收确认、超时重传等。流量控制采用停等机制，完成给定测试文件的传输。

二、协议设计

（一）报文格式

在本次实验中，仿照 TCP 协议的报文格式进行了数据报设计，其中整个报文包括报头段和数据段。报头包括源端口号、目的端口号、序列号、确认号、消息数据长度、标志位、校验值。其中标志位包括 FIN、PUSH、ACK、SYN 四位。数据段则是大小为size的数据。
具体的设置如下所示。



```
#define SYN 0b01
#define ACK 0b10
#define FIN 0b100
#define PUSH 0b1000

-----

struct Header {
    uint16_t sourcePort;
    uint16_t destinationPort;
    uint32_t seqnum;
    uint32_t acknum;
    uint16_t Flags;
    uint16_t checksum;
    uint32_t length;
};

struct message
{
    Header head;
    char data[size];
    message() :head{ 0, 0, 0, 0, 0, 0,0 } {
        memset(data, 0, size);
    }
}
```

(二) 消息传输机制

在消息传输中，我们完成了建立连接，超时重传，差错检测等功能。本次实验采用停等机制以及单方向传输，所以对于发送方，我们在收到确认的包之前持续进行等待。下面进行详细介绍。

1. 建立连接——三次握手

我们仿照TCP协议设置了三次握手，首先由发送方向接收方发送连接请求，并将FLAG设置为SYN，然后，接收方会发送报文设置FLAG为SYN以及ACK，并相应的加上acknum。最后发送方再发送一个ACK的包，从而确认收到。实现三次握手，连接建立。



```
D:\socket\lab3-client\x64\Del x D:\socket\lab3-1\x64\Debug x + v
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 0
Acknowledgment Number: 0
Flags: SYN
Checksum: 49364
Length: 0
-----
发送第二次握手的报文
-----
Source Port: 8090
Destination Port: 8080
Sequence Number: 0
Acknowledgment Number: 1
Flags: SYN ACK
Checksum: 49361
Length: 0
-----
收到第三次握手的报文
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 1
Acknowledgment Number: 1
Flags: ACK
Checksum: 49361
Length: 0
-----
握手成功

-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 0
Acknowledgment Number: 0
Flags: SYN
Checksum: 49364
Length: 0
-----
收到第二次握手的报文，第二次握手成功
-----
Source Port: 8090
Destination Port: 8080
Sequence Number: 0
Acknowledgment Number: 1
Flags: SYN ACK
Checksum: 49361
Length: 0
-----
发送第三次握手的报文
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 1
Acknowledgment Number: 1
Flags: ACK
Checksum: 49361
Length: 0
-----
```

2. 差错检测

对于差错检测，我们采用了校验和的形式，针对报文报文内容计算校验和，并将其设置的报文中。然后由接收方在收到时进行校验和的确认。因为校验和计算时需要16位，因此我们发送时需要做到发送的message的大小为16位的倍数。

3. 停等机制与接收确认

本次实验采用超时重传的机制，同时是单向传输。所以我们对于发送方设置只有当接收到对应的ACK报文时，才会发送下一个消息。接收方会记录发送方的发送的消息大小，并随着ACK包传回发送端，

从而让发送端进行判断ACK是否正确。正确之后才会继续发送下一个包。而对于接收方来说，我们设定其发送ACK包时，其seqnum不会变化。

4. 超时重传

(1) 数据包发送丢失

对于可能发生丢包的问题，设置了超时重传机制。当超过规定的 `Wait_Time` 仍没有收到对应的接收端发送的 ACK 确认报文，将重新发送数据，若过了相应时间仍旧没有收到ACK，则会继续发送，当重传超过规定次数，则会输出信息，并进行关闭。

```
-----
开始传输
扣一传文件，扣2断开连接
1
请输入你要传送的文件
1.jpg
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 1
Acknowledgment Number: 1
Flags: PUSH
Checksum: 51811
Length: 1857353
-----
已超时，现在进行重传
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 1
Acknowledgment Number: 1
Flags: PUSH
Checksum: 51811
Length: 1857353
-----
已超时，现在进行重传
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 1
```

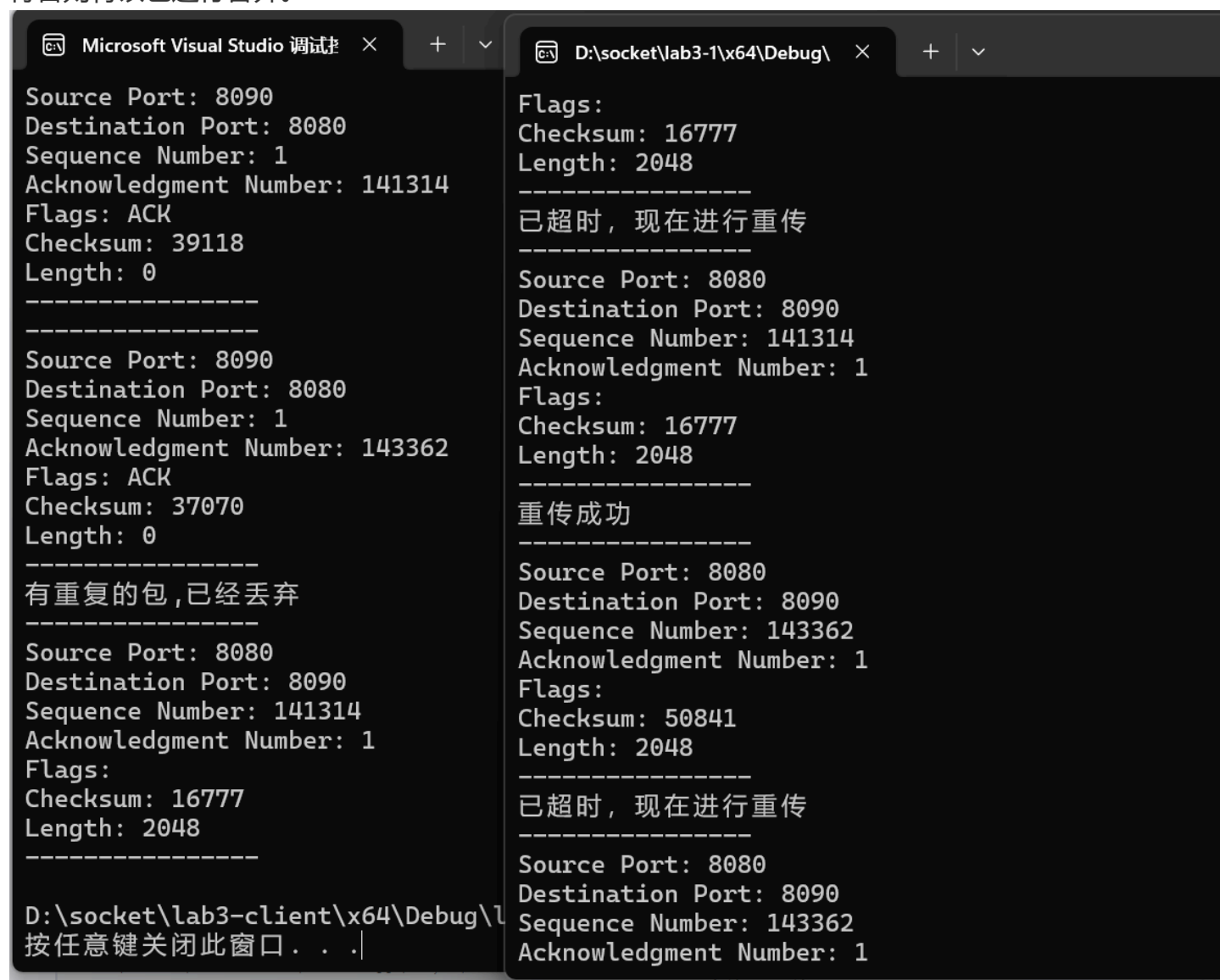
如图所示，当，超时未收到时，就会进行重传。

(2) 数据包接收丢失

当数据包接收丢失时，因为不会向发送方进行ACK确认，所以同样会在超时后在由发送端进行重传。

(3) 数据包失序

当接收或发送过程中发生延时问题时，可能会导致发送方发送重复的包，对于这样就需要接收端进行识别，因为acknum会和发送方的seqnum相对应，因此通过这种方法判断收到的包是否符合，如果不符合则将该包进行舍弃。



```
Microsoft Visual Studio 调试
Source Port: 8090
Destination Port: 8080
Sequence Number: 1
Acknowledgment Number: 141314
Flags: ACK
Checksum: 39118
Length: 0
-----
Source Port: 8090
Destination Port: 8080
Sequence Number: 1
Acknowledgment Number: 143362
Flags: ACK
Checksum: 37070
Length: 0
-----
有重复的包,已经丢弃
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 141314
Acknowledgment Number: 1
Flags:
Checksum: 16777
Length: 2048
-----
D:\socket\lab3-client\x64\Debug\1
按任意键关闭此窗口...|

D:\socket\lab3-1\x64\Debug\
Flags:
Checksum: 16777
Length: 2048
-----
已超时, 现在进行重传
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 141314
Acknowledgment Number: 1
Flags:
Checksum: 16777
Length: 2048
-----
重传成功
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 143362
Acknowledgment Number: 1
Flags:
Checksum: 50841
Length: 2048
-----
已超时, 现在进行重传
-----
Source Port: 8080
Destination Port: 8090
Sequence Number: 143362
Acknowledgment Number: 1
```

这里我们人为的设置了延时，从而引起了重传，可以看到当143362这个包收到时，重传的141314才刚到达，这是就会打印日志。在程序中，接收端会继续接收余下的包，这里是我们为了演示而进行的break。

5. 断开连接——四次挥手

我们仿照TCP的四次挥手设计了断开连接的操作。当发送端发送FIN想要断开时，接收方先回复ACK，再发送FIN，最后由发起方发送ACK.接受方收到ACK后立即关闭。发起方等待一段时间没有信息则自动关闭。seq及acknum的变化与握手时相似，不再赘述。

----- 发送第三次挥手的报文 ----- Source Port: 8090 Destination Port: 8080 Sequence Number: 2 Acknowledgment Number: 1857356 Flags: FIN ACK Checksum: 26981 Length: 0 ----- 收到第四次挥手的报文 ----- Source Port: 8080 Destination Port: 8090 Sequence Number: 1857356 Acknowledgment Number: 3 Flags: ACK Checksum: 26984 Length: 0 ----- 终于说再见 bye-bye	Length: 0 ----- 收到第三次挥手的报文 ----- Source Port: 8090 Destination Port: 8080 Sequence Number: 2 Acknowledgment Number: 1857356 Flags: FIN ACK Checksum: 26981 Length: 0 ----- 发送第四次挥手的报文 ----- Source Port: 8080 Destination Port: 8090 Sequence Number: 1857356 Acknowledgment Number: 3 Flags: ACK Checksum: 26984 Length: 0 ----- 等待时间已过，断开连接，挥手成功
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6. 状态机

(1) 发送端

- 建立连接，发送报文，Seq = x，启动计时器，等待回复
 - 超时未收到 ACK 确认报文：重新发送数据并重新计时
- 收到 ACK 确认报文，且 Ack 及相关标志位匹配成功：继续发送下一个报文或关闭连接
 - 如果接收到错误报文，则会输出并停止。

(2) 接收端

- 建立连接，等待接收
 - 收到报文，但 Seq 或相关标志位不匹配：丢弃报文，输出日志，继续等待
- 收到报文，且 Seq 或相关标志位匹配：接收报文，发送对应 Ack，继续等待下一个报文或关闭连接

三、代码实现

(一) 协议设计

对应标志位

```
#define SYN 0b01
#define ACK 0b10
#define FIN 0b100
#define PUSH 0b1000
```

我们将message分为Header和data两个区域。Header作为每个包的信息，而data则是要传输的数据。

```
struct message
{
    Header head;
    char data[size];
    message() :head{ 0, 0, 0, 0, 0, 0,0 } {
        memset(data, 0, size);
    }
    void setchecksum();
    bool check();
    void setsyn() { this->head.Flags |= SYN; }
    void setack() { this->head.Flags |= ACK; }
    void setfin() { this->head.Flags |= FIN; }
    void setpush() { this->head.Flags |= PUSH; }
    bool ispush() { return (this->head.Flags & PUSH) != 0; }
    bool issyn(){ return (this->head.Flags & SYN) != 0; }
    bool isack() { return (this->head.Flags & ACK) != 0; }
    bool isfin() { return (this->head.Flags & FIN) != 0; }
    void print() {
        printf("-----\n");
        printf("Source Port: %u\n",head.sourcePort);
        printf("Destination Port: %u\n",head.destinationPort);
        printf("Sequence Number: %u\n", head.seqnum);
        printf("Acknowledgment Number: %u\n",head.acknum);
        printf("Flags: ");
        if (issyn()) printf("SYN ");
        if (isfin()) printf("FIN ");
        if (head.Flags & ACK) printf("ACK ");
        if (head.Flags & PUSH) printf("PUSH ");
        printf("\n");
        printf("Checksum: %u\n", head.checksum);
        printf("Length: %u\n",head.length);
        printf("-----\n");
    }
};
```

在结构体中，我们设置了相应的校验函数，标志位设置，以及输出的函数。标志位的设置和检验如下所示。

```
void message::setchecksum()
{
    this->head.checksum = 0;
    uint32_t sum = 0;
    uint16_t* p = (uint16_t*)this;
    for (int i = 0; i < sizeof(*this) / 2; i++)
    {
        sum += *p++;
        while (sum >> 16)
        {
            sum = (sum & 0xffff) + (sum >> 16);
        }
    }
    this->head.checksum = ~(sum & 0xffff);
}

bool message::check()
{
    uint32_t sum = 0;
    uint16_t* p = (uint16_t*)this;
    for (int i = 0; i < sizeof(*this) / 2; i++)
    {
        sum += *p++;
        while (sum >> 16)
        {
            sum = (sum & 0xffff) + (sum >> 16);
        }
    }
    return (sum & 0xffff) == 0xffff;
}
```

(二) 初始化

我们在发送端和接受端采用相同的初始化方式，除了发送端设置为非阻塞模式，而因为是单方向传输，所以在接收端采用阻塞模式。下面以发送方为例。


```

bool initial()
{
    serversocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (serversocket == INVALID_SOCKET)
    {
        cout << "出错了" << ", 错误请看" << WSAGetLastError();
        WSACleanup();
        return false;
    }
    u_long mode = 1; // 1 表示非阻塞模式
    if (ioctlsocket(serversocket, FIONBIO, &mode) != NO_ERROR) {
        std::cerr << "无法设置非阻塞模式: " << WSAGetLastError() << std::endl;
        closesocket(serversocket);
        WSACleanup();
        return false;
    }
    sockaddr_in addrSrv;
    addrSrv.sin_family = AF_INET;
    const char* ip = "127.0.0.1";
    if (inet_pton(AF_INET, ip, &addrSrv.sin_addr) <= 0) {
        std::cerr << "inet_pton失败了 " << WSAGetLastError() << std::endl;
        closesocket(serversocket);
        WSACleanup();
        return false;
    }
    addrSrv.sin_port = htons(serverport);
    if(bind(serversocket, (SOCKADDR*)&addrSrv, sizeof(addrSrv)) == SOCKET_ERROR)
    {
        return false;
    }

    clientaddr.sin_family = AF_INET;
    clientaddr.sin_port = htons(clientport);
    if (inet_pton(AF_INET, ip, &clientaddr.sin_addr) <= 0) {
        std::cerr << "inet_pton失败了 " << WSAGetLastError() << std::endl;
        closesocket(serversocket);
        WSACleanup();
        return false;
    }
    return true;
}

```

(三) 建立连接——三次握手

1. 发送端

- 发送第一次握手消息，并开始计时，申请建立连接，然后等待接收第二次握手消息
 - 如果超时未收到，则重新发送，多次重传失败则握手失败。
- 收到正确的第二次握手消息后，发送第三次握手消息

```

bool connect()
{
    cout << "开始握手" << endl;
    message msg1;
    msg1.setsyn();
    msg1.head.seqnum = seq;
    int len = sizeof(clientaddr);
    if (send(msg1) == SOCKET_ERROR)
    {
        int error_code = WSAGetLastError();
        printf("sendto failed with error: %d\n", error_code);
        closesocket(serversocket); // 关闭套接字
        WSACleanup(); // 清理 Winsock
        return false;
    }
    cout << "发送第一次握手的报文" << endl;
    msg1.print();
    start_time = clock();
    int i = 0;
    message msg2;
    while (1)
    {
        if (recvfrom(serversocket, (char*)&msg2, sizeof(msg2), 0,
(SOCKADDR*)&clientaddr, &len) > 0)
        {
            if (!msg2.check() || msg1.head.seqnum + 1 != msg2.head.acknum || !
(msg2.issyn() && msg2.isack()))
            {
                cout << "有错误";
                return false;
            }
            else
            {
                cout << "收到第二次握手的报文，第二次握手成功" << endl;
                msg2.print();
            }
            break;
        }
        if (i > 2)
        {
            cout << "重传过多且失败,终止" << endl;
            i = 0;
            closesocket(serversocket); // 关闭套接字
            WSACleanup(); // 清理 Winsock
            return false;
        }
    }
}

```

```

    }
    end_time = clock();
    double elapsed_time = 1000.0 * (end_time - start_time) / CLOCKS_PER_SEC;
    if (elapsed_time > waittime)
    {
        start_time = clock();
        i++;
        cout << "已超时，现在进行重传" << endl;
        send(msg1);
    }

}

message msg3;
seq = seq + 1;
ack = msg2.head.seqnum + 1;
msg3.head.seqnum = msg1.head.seqnum + 1;
msg3.head.acknum = ack;
msg3.setack();
if (send(msg3) == SOCKET_ERROR)
{
    int error_code = WSAGetLastError();
    printf("sendto failed with error: %d\n", error_code);
    closesocket(serversocket); // 关闭套接字
    WSACleanup(); // 清理 Winsock
    return false;
}
cout << "发送第三次握手的报文" << endl;
msg3.print();
cout << sizeof(msg3) << endl;
return true;
}

```

2. 接收端

- 接收正确的第一次握手消息，发送第二次握手消息，并开始计时，等待接收第三次握手消息
- 接收到正确的第三次握手消息，连接成功建立
与发送方相似，不再附上代码。

(四) 数据传输

我们定义了send函数，便于进行发送。

```
int send(message &messg)
{
    messg.head.sourcePort = serverport;
    messg.head.destinationPort = clientport;
    messg.setchecksum();
    return sendto(serversocket, (char*)&messg, sizeof(messg), 0,
(SOCKADDR*)&clientaddr, sizeof(clientaddr));
}
```

1. 发送端

本次文件我们要进行文件传输，因此，我们需要找到对应的文件。这里采用输入文件名的形式，找到当前路径下的文件，然后进行传输。在传送文件名时，将文件的大小写入length。seq对应加一。标志位设置为push，用来标识发送的是文件名及大小。发送过程中同样采用超时重传。

```
int len = sizeof(clientaddr);
message file;
FILE* file1 = fopen(name, "rb");
if (!file1) {
    perror("无法打开文件");
    return ;
}

fseek(file1, 0, SEEK_END);
long fileSize = ftell(file1);
fseek(file1, 0, SEEK_SET);
strncpy(file.data, name, strlen(name));
file.head.seqnum = seq;
file.head.length = fileSize;
file.setpush();
file.head.acknum = ack;
seq +=1;
send(file);
file.print();
```

在收到确认后。执行文件传输操作。根据文件的大小以及每次发送的数据大小。来确定传输次数。同时每次采用超时重传策略。如果收到错误的报文则会报错终止。

```

int chunkSize = size;
    int sentBytes = 0;
    char buffer[size];
    int flag = 0;
    while (sentBytes < fileSize) {
        int readBytes = fread(buffer, 1, chunkSize, file1);
        message msg;
        msg.head.sourcePort = serverport;
        msg.head.destinationPort = clientport;
        msg.head.seqnum = seq;
        msg.head.acknum = ack;
        msg.head.length = readBytes;
        memcpy(msg.data, buffer, readBytes);
        msg.setchecksum();
        start_time = clock();
        if (sendto(serversocket, (char*)&msg, sizeof(Header) + size, 0,
(SOCKADDR*)&clientaddr, sizeof(clientaddr)) == SOCKET_ERROR) {
            int error_code = WSAGetLastError();
            printf("sendto failed with error: %d\n", error_code);
            fclose(file1);
            return false ;
        }
        msg.print();
        sentBytes += readBytes;
        seq += readBytes;
        message recmsg;
        while (1)
        {
            int len = sizeof(clientaddr);
            if (recvfrom(serversocket, (char*)&recmsg, sizeof(recmsg), 0,
(SOCKADDR*)&clientaddr, &len) > 0)
            {
                if (i != 0)
                    cout << "重传成功" << endl;
                i = 0;
                if (!recmsg.check() ||
!recmsg.isack() || recmsg.head.acknum != seq)
                {
                    cout << "有错误" << endl;
                    flag = 1;
                    break;
                }
                break;
            }
        }
        else
        {

```

```

        if (i > 2)
        {
            cout << "重传过多且失败,终止" << endl;
            closesocket(serversocket); // 关闭套接字
            WSACleanup(); // 清理 Winsock
            return false;
        }
        end_time = clock();
        double elapsed_time = 1000.0 * (end_time - start_time) /
CLOCKS_PER_SEC;

        if (elapsed_time > waittime)
        {
            start_time = clock();
            i++;
            cout << "已超时, 现在进行重传" << endl;
            sendto(serversocket, (char*)&msg, sizeof(Header) +
size, 0, (SOCKADDR*)&clientaddr, sizeof(clientaddr));
            msg.print();
        }
    }

    if (flag == 1)
        break;
}
cout << "文件传输成功" << endl;
fclose(file1);
return true;

```

2. 接收端

对于接收端，会根据发送报文的不同标志位进行处理。当收到的包为PUSH时，会根据内容创建新的的文件，同时存储文件大小。接着进入循环之中。当收到的文件完整时，关闭文件。如果收到FIN，则对应使用挥手函数。接收端还会对包进行判断，判断是否为无用的重复包等，若是则丢弃，并继续等待。

```

int len = sizeof(clientaddr);
    message msg;
    message sendmsg;
    ofstream outFile;
    long recived = 0;
    long filesize = 0;
    int i = 0;
    while (true)
    {

        recvfrom(serversocket, (char*)&msg, sizeof(msg), 0, (SOCKADDR*)&clientaddr,
&len);

        if (msg.ispush() && msg.check() && msg.head.acknum == seq && msg.head.seqnum == ack)
        {
            recived = 0;
            if (outFile.is_open()) {
                outFile.close();
            }

            outFile.open(msg.data, std::ios::out | std::ios::binary);
            if (!outFile) {
                perror("无法打开文件");
                break;
            }
            cout << "new file";
            filesize = msg.head.length;
            ack += 1;
            sendmsg.setack();
            sendmsg.head.acknum = ack;
            sendmsg.head.seqnum = seq;
            cout << filesize;
            send(sendmsg);
            sendmsg.print();

        }
        else if(msg.check() && msg.head.acknum == seq && (recived+msg.head.length)
<filesize && msg.head.seqnum == ack)
        {
            recived += msg.head.length;
            ack += msg.head.length;
            sendmsg.head.acknum = ack;
            sendmsg.head.seqnum = seq;
            sendmsg.setack();
            send(sendmsg);
            sendmsg.print();
            if (outFile.is_open())

```



```

        {
            outFile.write(msg.data, msg.head.length);
        }
    }
    else if (msg.check() && msg.head.acknum == seq && (recived +
msg.head.length) >= filesize && msg.head.seqnum == ack)
    {
        recived += msg.head.length;
        ack += msg.head.length;
        sendmsg.head.acknum = ack;
        sendmsg.head.seqnum = seq;
        sendmsg.setack();
        send(sendmsg);
        sendmsg.print();
        if (outFile.is_open())
        {
            outFile.write(msg.data, msg.head.length);
        }
        outFile.close();
        cout << "传输文件成功" << endl;
    }
    else if (msg.check() && msg.head.acknum == seq && msg.head.seqnum != ack)
    {
        cout << "有重复的包,已经丢弃" << endl;
        msg.print();
    }
    else if (msg.check() && msg.isfin())
    {
        ack += 1;
        cout << "收到第一次挥手的报文" << endl;
        msg.print();
        if (!saybye())
        {
            cout << "没能说再见" << endl;
        }
        else
        {
            cout << "bye bye" << endl;
            closesocket(serversocket);
            WSACleanup();
            break;
        }
    }
}
}

```

(五) 断开连接——四次挥手

1. 发送端

- 发送第一次挥手消息，并开始计时，提出断开连接，然后等待接收第二次挥手消息
 - 如果超时未收到，则重新发送
- 收到正确的第二次挥手消息后，等待接收第三次挥手消息
- 接收到正确的第三次挥手消息，输出日志，准备断开连接
- 再等待设定时间时间，确定没有消息传来，断开连接。

```

bool saybye()
{
    message msg1;
    msg1.setfin();
    msg1.head.seqnum = seq;
    seq++;
    int len = sizeof(clientaddr);
    if (send(msg1) == SOCKET_ERROR)
    {
        int error_code = WSAGetLastError();
        printf("sendto failed with error: %d\n", error_code);
        closesocket(serversocket); // 关闭套接字
        WSACleanup(); // 清理 Winsock
        return false;
    }
    cout << "发送第一次挥手的报文" << endl;
    msg1.print();
    start_time = clock();
    int i = 0;
    message msg2;
    while (1)
    {
        if (recvfrom(serversocket, (char*)&msg2, sizeof(msg2), 0,
(SOCKADDR*)&clientaddr, &len) > 0)
        {
            cout << msg2.head.acknum << endl;
            if (!(seq == msg2.head.acknum) || !msg2.isack() || !msg2.check())
            {
                cout << "有错误";
                return false;
            }
            cout << "收到第二次挥手的报文" << endl;
            msg2.print();
            break;
        }
        if (i > 2)
        {
            cout << "重传过多且失败,终止" << endl;
            i = 0;
            closesocket(serversocket); // 关闭套接字
            WSACleanup(); // 清理 Winsock
            return false;
        }
        end_time = clock();
        double elapsed_time = 1000.0 * (end_time - start_time) / CLOCKS_PER_SEC;
    }
}

```

```

        if (elapsed_time > waittime)
        {
            start_time = clock();
            i++;
            cout << "已超时，现在进行重传" << endl;
            send(msg1);
        }
    }
    ack++;
    message msg4;
    start_time = clock();
    while (1)
    {
        if (recvfrom(serversocket, (char*)&msg4, sizeof(msg4), 0,
(SOCKADDR*)&clientaddr, &len) > 0)
        {
            if (!(seq == msg2.head.acknum) || !msg4.isack() || !msg4.check() ||
!msg4.isfin())
            {
                cout << "有错误";
                return false;
            }
            cout << "收到第三次挥手的报文" << endl;
            msg4.print();

            break;
        }
        end_time = clock();
        double elapsed_time = 1000.0 * (end_time - start_time) / CLOCKS_PER_SEC;
        if (elapsed_time > 100)
        {
            cout << "等待时间过长，自动关闭" << endl;
            closesocket(serversocket); // 关闭套接字
            WSACleanup(); // 清理 Winsock
            return true;
        }
    }

    ack++;
    message msg3;;
    msg3.head.seqnum = seq;
    msg3.head.acknum = ack;
    msg3.setack();
    if (send(msg3) == SOCKET_ERROR)
    {
        int error_code = WSAGetLastError();
    }

```

```

        printf("sendto failed with error: %d\n", error_code);
    }
    cout << "发送第四次挥手的报文" << endl;
    msg3.print();
    start_time = clock();
    while (1)
    {
        if (recvfrom(serversocket, (char*)&msg4, sizeof(msg4), 0,
(SOCKADDR*)&clientaddr, &len) > 0)
        {
            cout << "挥手失败" << endl;
            break;
        }
        end_time = clock();
        double elapsed_time = 1000.0 * (end_time - start_time) / CLOCKS_PER_SEC;
        if (elapsed_time > 200)
        {
            cout << "等待时间已过，断开连接，挥手成功" << endl;
            closesocket(serversocket); // 关闭套接字
            WSACleanup(); // 清理 Winsock
            return true;
        }
    }
    return true;
}

```

2. 接收端

- 接收正确第一次挥手消息，发送第二次挥手消息，同意断开连接
- 发送第三次挥手消息，然后等待接收第四次挥手消息
- 接收到正确的第四次挥手消息，输出日志，断开连接

```

bool saybye()
{

    message msg[3];
    int len = sizeof(clientaddr);
    msg[0].setack();
    msg[0].head.seqnum = seq;
    msg[0].head.acknum = ack;
    if (send(msg[0]) == SOCKET_ERROR)
    {
        int error_code = WSAGetLastError();
        printf("sendto failed with error: %d\n", error_code);
    }
    cout << "发送第二次挥手的报文" << endl;
    msg[0].print();
    seq++;
    msg[1].setack();
    msg[1].setfin();
    msg[1].head.acknum = ack;
    msg[1].head.seqnum = seq;
    send(msg[1]);
    cout << "发送第三次挥手的报文" << endl;
    msg[1].print();
    seq++;
    while (1)
    {
        recvfrom(serversocket, (char*)&msg[2], sizeof(msg[2]), 0,
(SOCKADDR*)&clientaddr, &len);
        if (!msg[2].check() || !msg[2].isack() ||! (msg[2].head.acknum ==seq))
        {
            cout << "校验错误" << endl;
            return false;
        }
        break;
    }
    cout << "收到第四次挥手的报文" << endl;
    msg[2].print();
    cout << "终于说再见" << endl;

    return true;
}

```

四、传输测试与性能分析

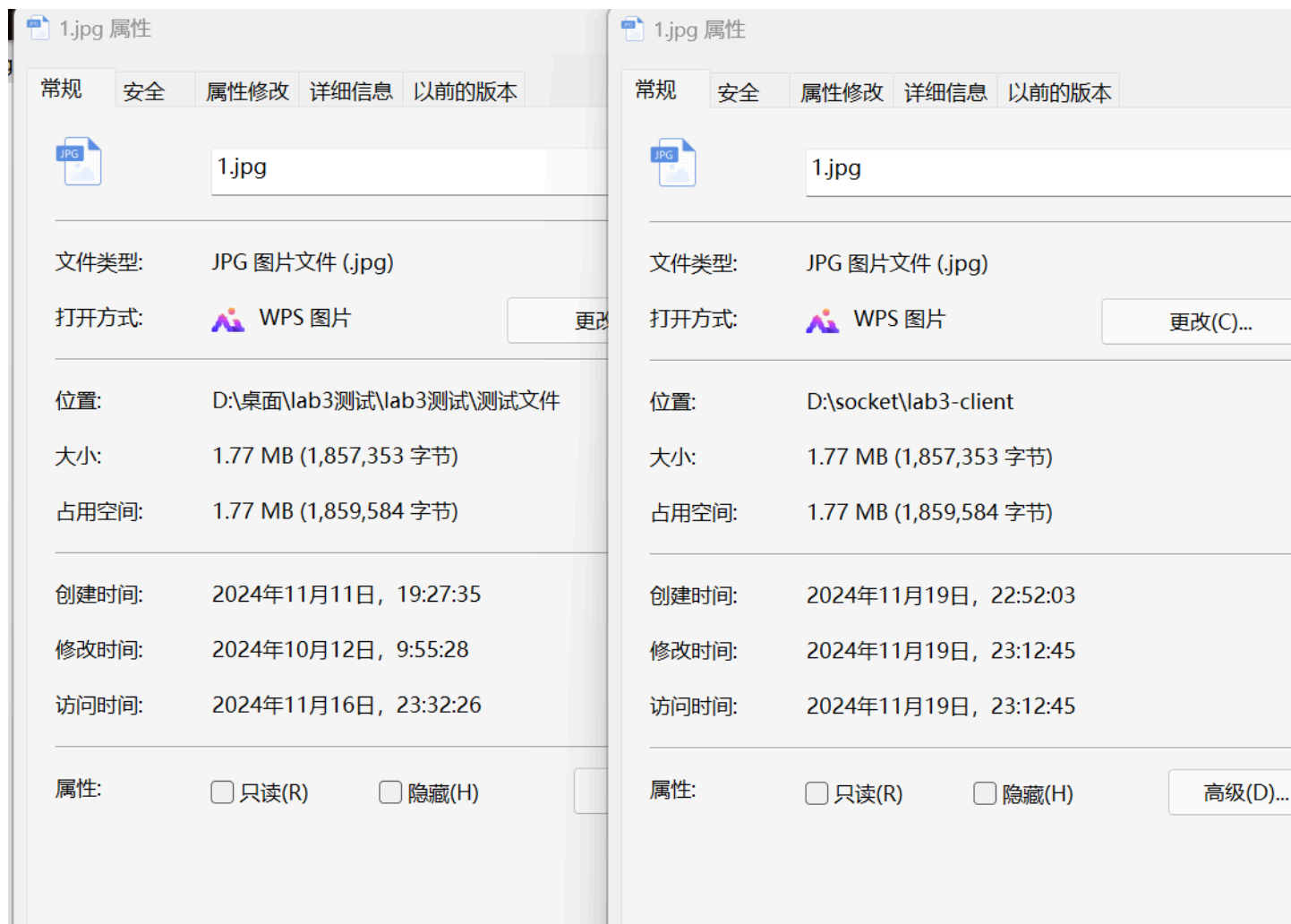
(一) 传输测试

1.传输测试

在这里测试了1.jpg的文件，设置丢包为0，延时为10ms.可以看到文件传输成功。在发送端输出了传输用时以及传输文件的大小。并输出了吞吐率。



接着查看文件的属性。可以看到两个文件的大小以及文件名一致。传输无误。



接着进行抓包，可以看到对应的抓包

(ip.addr == 127.0.0.1) and (udp.port == 8080)						
No.	Time	Source	Destination	Protocol	Length	Info
3	7.316822	127.0.0.1	127.0.0.1	UDP	2100	8080 → 8090 Len=2068
4	7.317312	127.0.0.1	127.0.0.1	UDP	2100	8090 → 8080 Len=2068
5	7.317592	127.0.0.1	127.0.0.1	UDP	2100	8080 → 8090 Len=2068
20	17.369282	127.0.0.1	127.0.0.1	UDP	2100	8080 → 8090 Len=2068
21	17.370388	127.0.0.1	127.0.0.1	UDP	2100	8090 → 8080 Len=2068
22	17.370475	127.0.0.1	127.0.0.1	UDP	2100	8080 → 8090 Len=2068
23	17.370665	127.0.0.1	127.0.0.1	UDP	2100	8090 → 8080 Len=2068
24	17.370841	127.0.0.1	127.0.0.1	UDP	2100	8080 → 8090 Len=2068
25	17.371009	127.0.0.1	127.0.0.1	UDP	2100	8090 → 8080 Len=2068
26	17.371165	127.0.0.1	127.0.0.1	UDP	2100	8080 → 8090 Len=2068
27	17.371271	127.0.0.1	127.0.0.1	UDP	2100	8090 → 8080 Len=2068

2.丢包重传测试

这里我们设置丢包率为百分之五，可以通过路由器的日志看到发生了miss。但最后 仍然传输成功。

Source Port: 8090
Destination Port: 9000
Sequence Number: 1
Acknowledgment Number: 5896194
Flags: ACK
Checksum: 50398
Length: 0

Source Port: 8090
Destination Port: 9000
Sequence Number: 1
Acknowledgment Number: 5898242
Flags: ACK
Checksum: 48350
Length: 0

Source Port: 8090
Destination Port: 9000
Sequence Number: 1
Acknowledgment Number: 5898507
Flags: ACK
Checksum: 48085
Length: 0

传输文件成功

Sequence Number: 5894146
Acknowledgment Number: 1
Flags:
Checksum: 55363
Length: 2048

Source Port: 8080
Destination Port: 9000
Sequence Number: 5896194
Acknowledgment Number: 1
Flags:
Checksum: 43529
Length: 2048

Source Port: 8080
Destination Port: 9000
Sequence Number: 5898242
Acknowledgment Number: 1
Flags:
Checksum: 18378
Length: 265

文件传输成功
传输用时107672ms
传输文件大小5898505字节
吞吐率为54.7822字节/ms
扣一传文件，扣2断开连接

Router

路由器IP: 127.0.0.1 服务器IP:
端口: 9000 服务器端
丢包率: 5% 延时:
确定
日志
count:18.
count:19.
Miss a packet.
count:1.
count:2.
count:3.
count:4.
count:5.
count:6.
count:7.
count:8

接着我们再查看一下文件传输后的属性。可以看到依旧没有失误。

2.jpg 属性

常规 安全 属性修改 详细信息 以前的版本

JPG

2.jpg

文件类型: JPG 图片文件 (.jpg)
打开方式: WPS 图片
位置: D:\桌面\lab3测试\lab3测试\
大小: 5.62 MB (5,898,505 字节)
占用空间: 5.62 MB (5,902,336 字节)
创建时间: 2024年11月11日, 19:27:35
修改时间: 2024年10月12日, 9:55:27
访问时间: 2024年11月16日, 23:32:26
属性: ☐ 只读(R) ☐ 隐藏(H)

2.jpg 属性

常规 安全 属性修改 详细信息 以前的版本

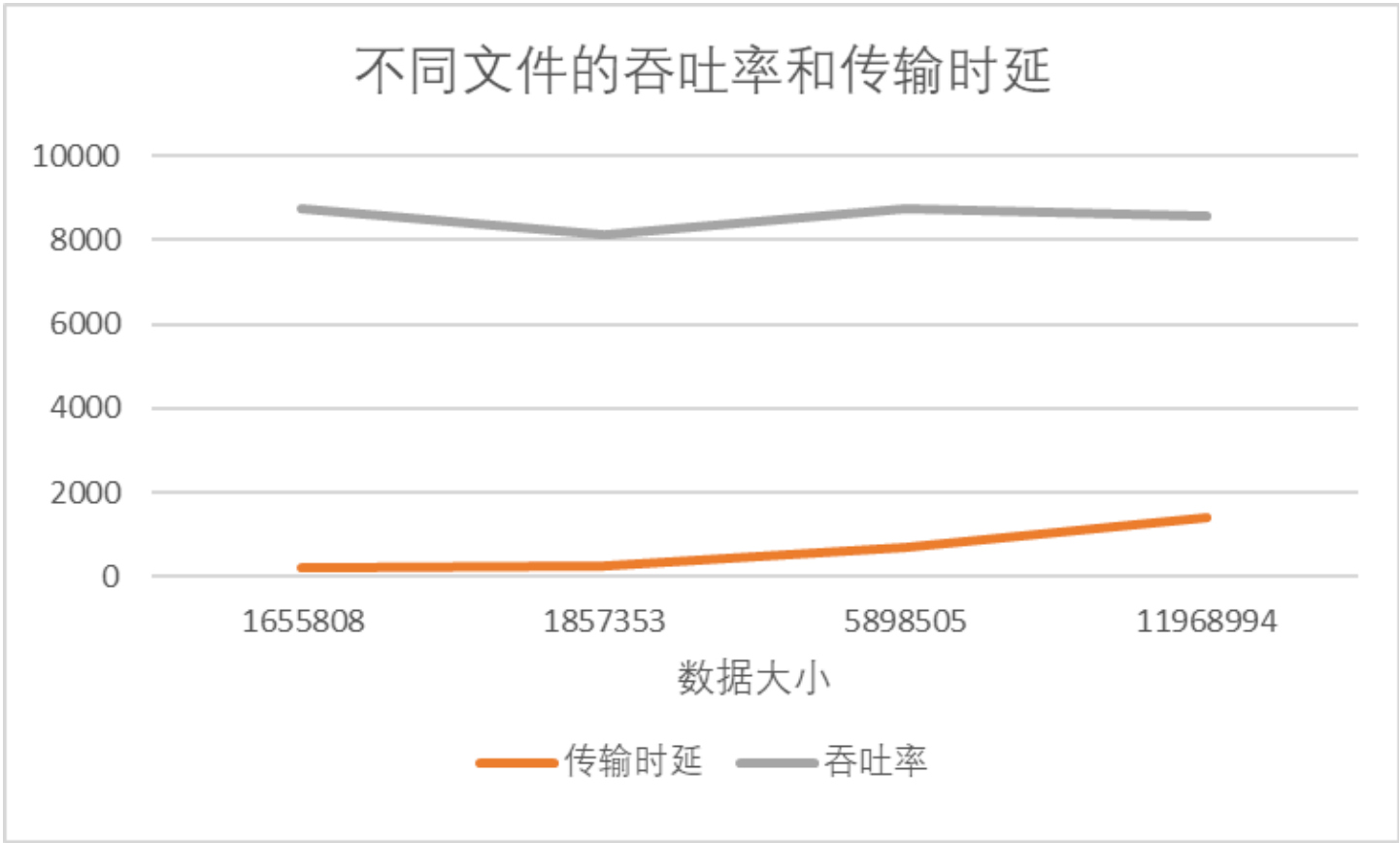
JPG

2.jpg

文件类型: JPG 图片文件 (.jpg)
打开方式: WPS 图片
位置: D:\socket\lab3-client
大小: 5.62 MB (5,898,505 字节)
占用空间: 5.62 MB (5,902,336 字节)
创建时间: 2024年11月14日, 22:43:44
修改时间: 2024年11月20日, 23:35:29
访问时间: 2024年11月20日, 23:38:41
属性: ☐ 只读(R) ☐ 隐藏(H)

(二) 性能分析

接下来我们统计了所给测试文件进行传输时的吞吐率以及传输时延。得到下面的对应图表。



可以看到随着文件大小，传输时间会对应增加。而对于吞吐率，则维持在一个相对稳定的范围。

五、问题反思

(一) 设置和检验校验和的位数

在计算校验和的时候涉及到16位的位移。需要计算时是16的倍数。在传输文件时，会存在不是16倍数的情况。后续采取将发送的都补全为16的倍数。

(二) 建立连接时的第三次握手

当进行握手时，可能会存在第三次握手的包没收到的情况。这时发送方认为成功，但接收端不会发回进行确认。存在连接不建立的情况。