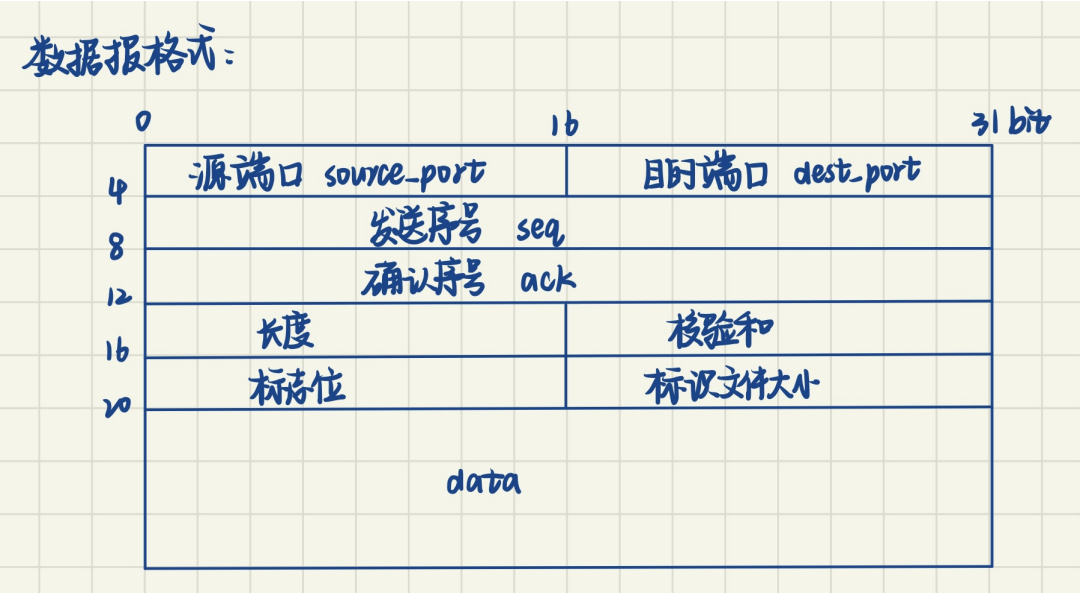
Lab3-1 实现UDP可靠数据传输

2010239 李思凡

一、数据报格式

基于UDP数据报格式来设计本实验中传输的数据报格式，添加了发送序列号、确认序列号、标志位和标识文件大小，如下图所示。



使用结构体来存储数据报的格式，整个数据报占8192字节，data段占8172字节。

```
        // 定义消息格式
        struct Message {
            unsigned short source_port = 0;
            unsigned short dest_port = 0;
            int seq = 0;
            int ack = 0;
            unsigned short lenth = 0;    //data的长度
            unsigned short checksum = 0;
            unsigned short flag = 0;
            unsigned short index = 0;    //传完整个文件需要的数据报个数（1个index=0,2个
index=1...）
            char data[DSIZE] = { 0 };
        };
```

其中flag是一些标志位，从低位到高位分别为FIN，SYN，ACK，REP，SF，EF，分别表示“断开连接信息”，“建立连接信息”，“ACK确认号有效”，“重复发送的信息”，“文件头信息”，“文件的最后一条数据报”。接收数据报后根据标志位判断消息类型，分情况进行处理。

二、协议设计

（一）建立连接与断开连接

建立连接时参考TCP协议的三次握手。由于本实验中实现的为单向传输，即只有发送端主动向接收端发送消息，因此三次握手可以简化为两次。

（1）发送端向接收端发送一条[SYN]消息。该消息的SYN标志位为1，序列号为当前序列号。发送时采用确认重传式发送。

（2）接收端收到后返回一条[SYN, ACK]消息。接收端需先判断收到的消息是否正确，之后返回SYN和ACK标志位均为1的消息。

（3）发送端收到正确的[SYN, ACK]消息后表示连接建立成功。由于确认重传机制，未收到正确的会等待超时重传，重新发起建连。

断开连接时参考TCP的四次挥手。与握手同理，本实验中四次挥手可以简化为两次。

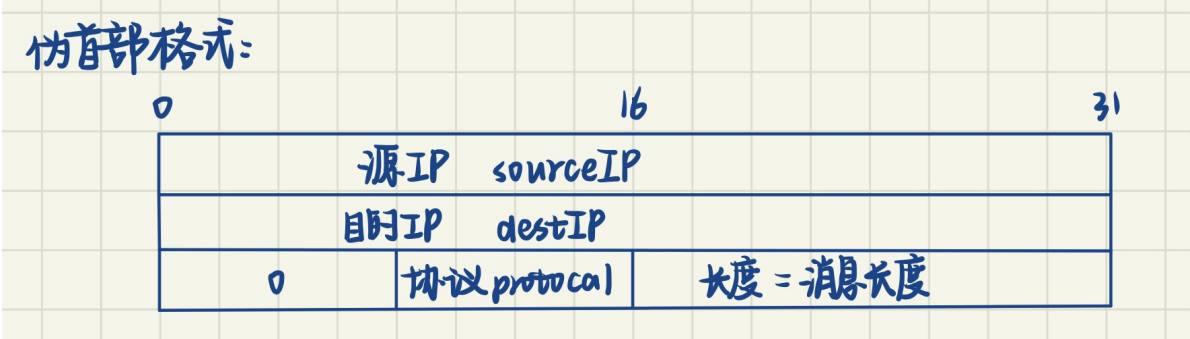
（1）发送端向接收端发送一条[FIN]消息。该消息的FIN标志位为1，序列号为当前序列号。发送时采用确认重传式发送。

（2）接收端收到后返回一条[FIN, ACK]消息。接收端需先判断收到的消息是否正确，之后返回FIN和ACK标志位均为1的消息，关闭socket。

（3）发送端收到正确的[FIN, ACK]消息后表示连接关闭成功，关闭socket。由于确认重传机制，未收到正确的会等待超时重传，重新发起断连。

（二）差错检测

由于本实验中需要进行数据校验，为了检验UDP数据报是否到达真正的目的地，设置伪首部，记录源IP和目的IP，以及协议和消息长度。发送端和接收端分别产生自己的伪首部，进行校验。



也采用结构体存储伪首部。

```
//定义伪首部
struct Pseudoheader
{
    unsigned long sourceIP{};
    unsigned long destIP{};
    char zero = 0;
    char protocal = 17; //udp协议号
    unsigned short lenth = sizeof(struct Message);
};
```

设置校验和：

（1）产生伪首部，校验和段清零，数据报用0补成16位整数倍；

（2）将伪首部和数据报一起看成16位整数序列；

（3）对各个域段进行**反码运算**求和，结果**取反**写入校验和段。

在每次发送消息前，将计算好的校验和填入校验和域段，随数据报一同发送。

检验校验和：

（1）产生伪首部，数据报用0补成16位整数倍；

（2）反码求和运算；

（3）若计算结果**全为1**，没有错误；否则，说明数据报有差错。

在每次收到消息时，先检验校验和是否正确，若校验和错误代表消息有错误，不能返回正确的ACK，期待重新收到发送方的正确消息。

（三）确认重传

采用rdt3.0的设计，下层通道既可能有差错也可能有丢失。由于存在丢失，需要设置序列号，在停等机制下序列号只需要1位，0和1来回交替。在ACK中携带确认分组的序列号，来判断接受端是否收到正确的分组。由于可能存在数据丢失，因此采用超时重传机制，若超过给定时间还未收到正确的ACK和数据报，则发送端重传当前分组。

采用确认重传机制后，发送端和接收端的状态转换如下：

1. 发送端：（假设当前序列号为0，序列号为1时同理）

   （1）上层调用send函数后，发送端打包数据、校验和、序列号（0），发送给接收端，同时启动定时器，进入等待ACK0的状态；

   （2）若从接收端收到ACK0且校验和正确，则说明数据传输无误，关闭定时器，更新当前序列号为1，退出执行后续内容。

   （3）若从接收端收到ACK1或校验和有误或未收到消息，说明数据传输有误，不进行操作，继续等待接收消息；

   （4）若等待时间超过设定的最大等待时间，则重新发送该消息，发送后继续等待对方回复。若重发次数超过最大重发次数，则断开连接。

2. 接收端：（假设当前序列号为0）

   （1）若接收到序列号为0的消息且校验和正确，则提取数据，接收端打包ack（为当前序列号0）和校验和，发送给发送端，更新序列号。

   （2）若接收到序列号为1的消息或校验和有误，则打包ack（非当前序列号1）和校验和，发送给发送端。

三、代码细节

发送端：

（一）发送端确认重传式发送

```
int sendPackage(Message_C message_C) {
    Message* message = message_C.message;
    printMessage(message_C);
    int result = -1;
    //第一次发送消息
    if (!isLoss()) {
        result = sendto(clientSocket, (char*)message, sizeof(struct Message), 0,
(struct                    sockaddr*)&serverAddrIn, sizeof(SOCKADDR));
    }
```

```
    if (result == -1) {
        cout << "Send Failed! " << endl;
    }
    cout << "Send Success! " << endl;
    clock_t start;
    clock_t end;
    // 开启定时器，等待ACK
    start = clock();
    int sendCount = 0;
    while (true) {
        Message recvMesg;
        int serverLength = sizeof(SOCKADDR);
        int recvLength = recvfrom(clientSocket, (char*)&recvMesg, sizeof(struct
Message), 0, (struct sockaddr*)&serverAddrIn, &serverLength);
        if (recvLength > 0) {
            Message_C recv_C(&recvMesg);
            printMessage(recv_C);
            //检查校验和与ACK
            //校验和正确且ACK正确，说明发送成功
            if (recv_C.isCk(&recvPheader) && recvMesg.ack == curseq) {
                end = clock();
                //更新序列号
                curseq = (curseq + 1) % 2;
                cout << "[ACK] : Package (SEQ:" << recvMesg.ack << ") send
success! " << endl;
                return 0;
            }
            //校验和错误或ACK错误，继续等待，直到超时重传
            else {
                end = clock();
                cout << "[RESEND] : Client received failed. Wait for timeout to
resend" << endl;
                if (sendCount > Max_Send_Count) {
                    cout << "[ERROR] : Resend too many times! " << endl;
                    return -1;
                }
                if ((end - start)/ CLOCKS_PER_SEC > Max_Wait_Time) {
                    sendCount++;
                    cout << "[Timeout] : Resend Package! " << endl;
                    result = -1;
                    if (!isLoss()) {
                        result = sendto(clientSocket, (char*)message,
sizeof(struct Message), 0, (struct sockaddr*)&serverAddrIn, sizeof(SOCKADDR));
                    }
                    if (result == -1) {
                        cout << "Send Failed! " << endl;
                    }
                    cout << "Send Success! " << endl;
                    printMessage(message_C);
                    start = clock(); //重置定时器
                }
            }
        }
        else {
            ......
            //此处省略，具体代码与上方else"校验和或ACK错误"相同。
        }
    }
```

```
}
```

（二）建立连接

```
int connection() {
    cout << "Wait for connection. " << endl;
    Message* message_SYN;
    message_SYN = new Message();
    message_SYN->source_port = clientPort;
    message_SYN->dest_port = serverPort;
    message_SYN->seq = curseq;
    message_SYN->ack = 0;
    Message_C message(message_SYN);
    message.setLen(0);
    message.setSYN(); //SYN标志位置1
    message.setcksum(&sendPheader);
    int result = sendPackage(message); //调用确认重传式发送
    if (result == -1) {
        cout << "[ERROR] : Connection Failed! " << endl;
        return -1;
    }
    cout << "[SUCC] : Connection Success! " << endl;
    return 0;
}
```

断开连接与建立连接类似，只是消息的FIN位置1。

（三）发送文件

首先读取文件，将每个文件的名称和大小记录在结构体中，依次发送每个文件，记录时间和吞吐率。

对单个文件的发送代码如下所示：

（1）首先发送文件头信息，它包括了文件的名称和文件的大小。其中对消息的设置主要体现在设置文件头标志位SF和设置数据为文件名称和大小；同时需要依据文件的大小和每条数据报的大小，计算出发完这个文件一共需要的数据报数目，填在index中。

（2）接着遍历需要的数据报个数，将文件内容按顺序填充的各个数据报中。如果已经填到了最后一个数据报，则该数据报的EF标志位置为1，便于接收端接收文件时判断结束。依次发送各个数据报。

```
int sendFile(File_C file) {
    //文件开始消息，包括文件名和大小
    Message* message_SEND;  message_SEND = new Message();
    message_SEND->source_port = clientPort; message_SEND->dest_port =
serverPort;
    message_SEND->seq = curseq; message_SEND->ack = 0;
    Message_C message_S(message_SEND);
    message_S.setACK(); message_S.setSF(); //设置为文件头
    message_S.setLen(sizeof(struct File_C));    message_S.setData((char*)&file);
    //计算需要多少个数据报才能发完一个文件
    int index = ceil(((double)file.size) / DSIZE);
    message_S.setIndex((short)index - 1); //index从0开始
    message_S.setcksum(&sendPheader);
    int result = sendPackage(message_S);
    if (result == -1) {
        cout << "[ERROR] : File " << file.name << " Send Failed! " << endl;
        return -1;
```

```
    }
    //发送文件内容
    ifstream fileStream(file.name, ios::binary | ios::app);
    int len = file.size;
    for (int i = 0; i < index; i++) {
        //读取文件内容
        char fileContent[DSIZE];
        fileStream.read(fileContent, fmin(len, DSIZE));
        //发送文件内容
        Message* message_FILE;  message_FILE = new Message();
        message_FILE->source_port = clientPort; message_FILE->dest_port =
serverPort;
        message_FILE->seq = curseq; message_FILE->ack = 0;
        Message_C message_F(message_FILE);  message_F.setACK();
        message_F.setLen(fmin(len, DSIZE)); message_F.setData(fileContent);
        if (i == message_S.getIndex()) {
            message_F.setEF();
        }
        message_F.setcksum(&sendPheader);
        len -= DSIZE;
        cout << "[FILE INDEX " << i << " in " << index << "]" << endl;
        int result = sendPackage(message_F);
        if (result == -1) {
            cout << "[ERROR] : File " << file.name << "Index " << i << " Send
Failed! " << endl;
            return -1;
        }
    }
}
```

接收端：

接收端相对简单，主要就是while循环不停接收消息，根据接收到的消息的标志位判断不同的消息，进行相应的处理。若校验和与序列号正确，则会返回正确的ACK并接收数据；否则就返回错误的ACK，即非当前序列号。在正确接收时，相应处理如下：

（一）接收到SYN消息

```
if (recv_C.isSYN()) {
    Message* message_SA = new Message();
    message_SA->source_port = serverPort;   message_SA->dest_port = clientPort;
    message_SA->seq = curseq;
    message_SA->ack = curseq;   //设置ack为当前序列号
    Message_C message_SAC(message_SA);
    message_SAC.setACK(); //设置ACK
    message_SAC.setSYN(); //设置SYN
    message_SAC.setLen(0);
    message_SAC.setcksum(&sendPheader);
    Message* message = message_SAC.message;
    cout << "Send SYN_ACK:" << endl;
    printMessage(message_SAC);
    int result = -1;
    if (!isLoss()) {
        result = sendto(serverSocket, (char*)message, sizeof(struct Message), 0,
(struct sockaddr*)&clientAddrIn, sizeof(SOCKADDR));
    }
    if (result == -1) {
```

```
        cout << "Send SYN_ACK Failed! " << endl;
        return -1;
    }
    cout << "Send SYN_ACK Success! " << endl;
    return 0;
}
```

接收到FIN消息同理，当接收正确时关闭服务器端socket即可。

接收文件消息后返回ACK也是同理。

（二）接收到关于文件的消息

接收到关于文件的消息有三种情况，第一是收到文件头消息，此时根据文件头指出的文件名，打开文件；第二种是收到关于文件内容的消息，此时向文件种写入数据；第三是收到的消息标志是文件的末尾，此时写入数据后表示文件接收成功，关闭文件。三种情况经过数据处理后都要返回相应的ACK消息。

```
//是文件头，识别信息，打开文件
if (recv_C.isSF()) {
    File_C fileC {};
    memcpy(&fileC, recvMesg.data, sizeof(struct File_C));
    cout << "[Receive file header]: Name:" << fileC.name << " Size:" <<
fileC.size << endl;
    fileSize = fileC.size;
    fileName = fileC.name;
    file.open(fileName, ios::out | ios::binary);
    //返回ACK
    sendACK(recvMesg.seq);
}
//不是文件头，向对应文件中写
else {
    file.write(recvMesg.data, recv_C.getLen());
    if (recv_C.isEF()) {
        cout << "[SUCC] : File " << fileName << "Receive Success! " << endl;
        file.close();
    }
    //返回ACK
    cout << "recv.seq=" << recvMesg.seq << endl;
    sendACK(recvMesg.seq);
}
```

四、演示

（一）建立连接

发送端：

可以看到，发送端发送一条SYN=1的消息，收到了接收端返回的SYN=1，ACK=1的消息。连接建立成功。

```
WSASTARTUP Success!
Client Socket Created!
Bind Success!
Wait for connection.
[Package]: (FIN: 0), (SYN: 1), (ACK: 0), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33336)
Send Success!
[Package]: (FIN: 0), (SYN: 1), (ACK: 1), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33332)
[ACK] : Package (SEQ:0) send success!
[SUCC] : Connection Success!
```

接收端：

可以看到，接收端正确收到了发送端的消息，并返回SYN=1, ACK=1的消息。

```
WSASTARTUP Success!
Server Socket Created!
Bind Success!
Receive Message:
[Package]: (FIN: 0), (SYN: 1), (ACK: 0), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33336)
Send SYN_ACK:
[Package]: (FIN: 0), (SYN: 1), (ACK: 1), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33332)
Send SYN_ACK Success!
```

（二）传输文件

文件头发送：

发送端：

可以看到，发送端发送了文件./test/1.jpg，其大小为1857353字节，SF=1。当前序列号为1，收到了ack=1的消息，说明当前消息发送成功。

```
Send file: ./test/1.jpg size: 1857353 begin!
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 1), (EF: 0), (seq: 1), (ack: 0), (len: 44), (cks: 33046)
Send Success!
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 1), (len: 0), (cks: 33332)
[ACK] : Package (SEQ:1) send success!
```

接收端：

可以看到，接收端收到了发送端的消息，并解析出文件的名称和大小，返回ack=1的确认消息。

```
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 1), (EF: 0), (seq: 1), (ack: 0), (len: 44), (cks: 33046)
[Receive file header]: Name:./test/1.jpg Size:1857353
ack=1
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 1), (len: 0), (cks: 33332)
Send ACK Success!
```

文件内容发送：

以发送最后一条数据报为例展示：

发送端：

可以看出，最后一条的文件index为227（0-227，共228个），此时EF被设置为1，当前seq为1。收到回复的ack也为1，表示最后一条发送成功。打印出发送所用时间为7308ms，平均吞吐率为254.153KB/s。

```
[FILE INDEX] 227 in 228
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 1), (seq: 1), (ack: 0), (len: 2309), (cks: 30992)
Send Success!
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 1), (len: 0), (cks: 33332)
[ACK] : Package (SEQ:1) send success!
Send file: ./test/1.jpg size: 1857353 in 7308ms with throughoutput in 254.153KB/s !
```

接收端：

可以看出，接收端接收到了EF=1的数据报，并显示成功接收，返回ack=1的数据报。

```
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 1), (seq: 1), (ack: 0), (len: 2309), (cks: 30992)
[SUCC] : File ./test/1.jpgReceive Success!
recv.seq=1
ack=1
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 1), (len: 0), (cks: 33332)
Send ACK Success!
```

（三）超时重传

由于设置了一定的丢包率，当超过设定时间（5s）后会重新发送。以其中一个重传事件为例。

发生错误：

接收端：

如图，接收端收到cks=25161的消息后，返回ack=1给发送端，但该返回消息发送失败。

```
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 0), (len: 8172), (cks: 25161)
recv.seq=1
ack=1
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 1), (len: 0), (cks: 33332)
Send ACK Failed!
```

发送端：

如图，发送端没有收到cks=25161的消息的确认ack，因此开始进入等待。

```
[FILE INDEX] 1 in 228
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 0), (len: 8172), (cks: 25161)
Send Success!
[RESEND] : Client received failed. Wait for timeout to resend
Time: 2 ms
[RESEND] : Client received failed. Wait for timeout to resend
Time: 18 ms
[RESEND] : Client received failed. Wait for timeout to resend
Time: 34 ms
[RESEND] : Client received failed. Wait for timeout to resend
Time: 50 ms
[RESEND] : Client received failed. Wait for timeout to resend
Time: 65 ms
[RESEND] : Client received failed. Wait for timeout to resend
Time: 81 ms
```

重传：

发送端：

如图，发送端等待超过5s后，开始重新发送消息。可以看到它重新发送了cks=25161的消息，并收到了相应的ack回复，表示发送成功。

```
Time: 5959 ms
[RESEND] : Client received failed. Wait for timeout to resend
Time: 5974 ms
[RESEND] : Client received failed. Wait for timeout to resend
Time: 5989 ms
[RESEND] : Client received failed. Wait for timeout to resend
Time: 6005 ms
[Timeout] : Resend Package!
Send Success!
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 0), (len: 8172), (cks: 25161)
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 0), (ack: 1), (len: 0), (cks: 33333)
[ACK] : Package (SEQ:1) send success!
```

接收端：

如图，接收端再次收到后又返回ack，此次ack发送成功。

```
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 0), (len: 8172), (cks: 25161)
ack=1
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 0), (ack: 1), (len: 0), (cks: 33333)
Send ACK Success!
```

（四）断开连接

发送端：

可以看出，发送端发送了FIN=1的断连消息，并收到相应的ack，关闭连接。

```
[Package]: (FIN: 1), (SYN: 0), (ACK: 0), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33337)
Send Success!
[Package]: (FIN: 1), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33333)
[ACK] : Package (SEQ:0) send success!
[SUCC] : Connection Destroyed!
```

接收端：

接收端收到FIN=1的消息，并返回FIN=1，ACK=1的确认消息。

```
Receive Message:
[Package]: (FIN: 1), (SYN: 0), (ACK: 0), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33337)
Send FIN_ACK:
[Package]: (FIN: 1), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33333)
Send FIN_ACK Success!
```