计算机网络实验3-1实验报告-UDP可靠数据传输

学号：2012039 姓名：冯朝芃

实验概要

本实验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，实现了类似TCP的**三次握手、四次挥手**、**校验和**的差错检测、**RDT3.0**可靠传输算法。

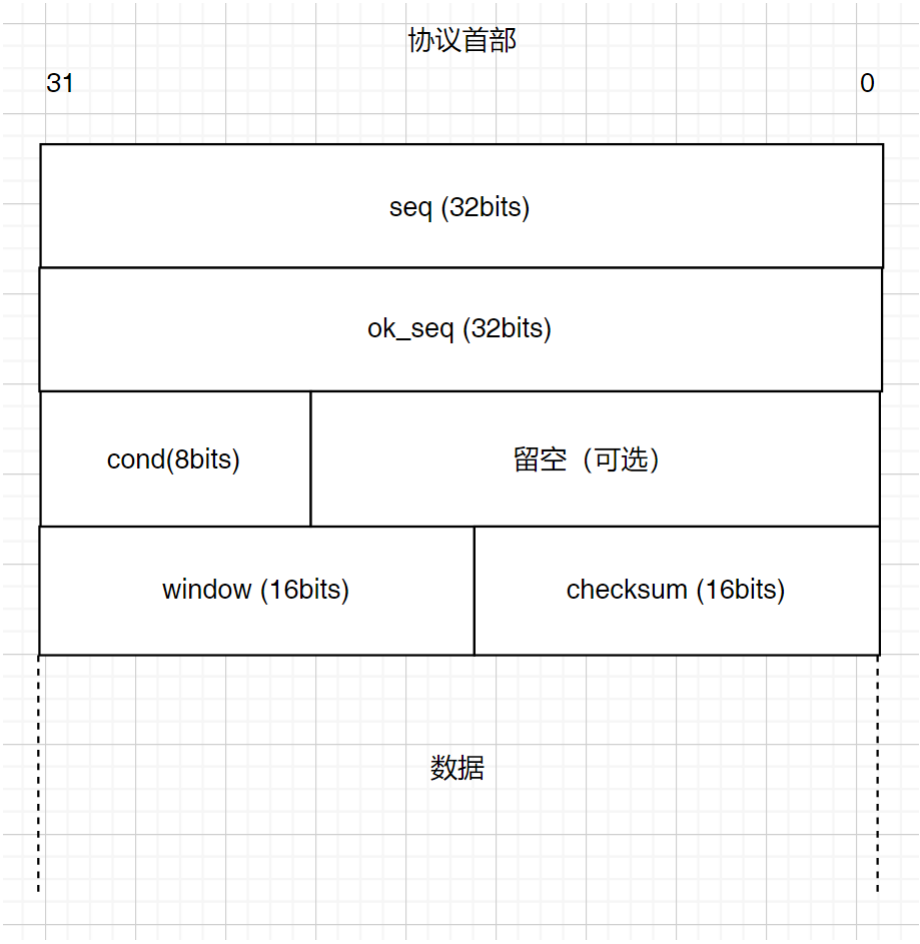
本实验流量控制采用停等机制，能够完成全部给定测试文件的传输。

协议设计

实验设计分为网络层和文件处理层，分层能够使我们的设计清晰简单。

网络层首部：

网络层首部模仿TCP首部设计，其中序列号长度为32bit，表示字节数位置。具体设计见下图。



seq和ok_seq：序列号和确认号，支持32bit长度，在接收和发送时根据发送和确认的字节位置进行填写。

cond：状态值，目前实现有ack、syn、fin，分别使用此八位的第0、1、2位是否设置来表示。

window：发送窗口通告，这里用不上。

checksum：校验和

这个数据结构在代码中称transmission。

文件处理层首部：

文件处理层同样具有一个协议首部。在传输时，在数据最开始包含有文件处理层首部的数据包将率先被接收方收到，接收方解析其中的内容，将**知晓需要接收多少个数据包。**

设计如下：

```
struct fileInfo{
    char name[256]; //文件名
    char type[8];   //文件类型
    unsigned int size;  //文件大小
};
```

协议时序

网络层：

与TCP协议不同，本实验seq将带有本次发送后，发送序列的结束字节位置，这样接收端就可以据此和保存的上一次的seq做减法来计算接收到的字节数量。ok_seq是对对方发送字节位置的确认，与TCP协议一致，是期望接收到的下一个字节位置，接收方可以将对方发来的seq增加1来得到ok_seq。

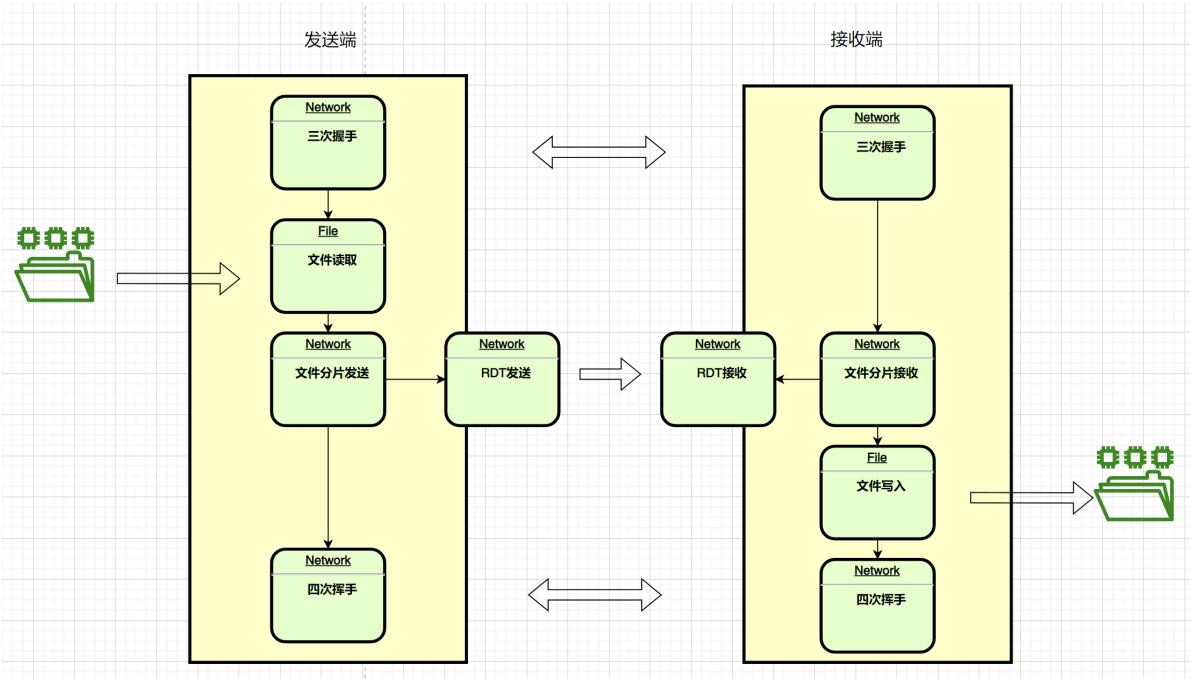
cond根据当前情况设置。checksum采用标准方法，发送端清零、计算、填写，接收端清零、计算、核对。

文件传输层：

fileInfo结构提在读取文件时被填写，接收端首先接收到该结构体，使用其中信息填写其他内部数据结构，并基于此决定接受次数和进行写盘。

功能模块划分

功能模块划分和工作基本流程见下图



文件层模块：

文件读取：为了能够读取较大的文件、更好的适应未来的流水线机制以及可能的对文件读取和发送并行化的实现，我们设计了一个文件分片链表的数据结构（blockHead）。链表上每个分片（unifiedBlock）将具有一个大小有上限的buffer并保存其大小。

文件读取模块将获取文件的基本信息：名称、类型和大小，使用其来填写fileInfo结构体。之后将这个结构体以及紧接的文件二进制内容存储进blockHead链表，这个过程中分片动态产生，每个分片大小不超过FILE_BLOCK_LIMIT（4096字节）。

文件写入：当网络层将第一个文件分片交给文件写入函数时，文件写入函数将首先解析fileInfo以获得文件信息。之后将遍历整个链表将文件写入。

网络层模块：

三次握手：三次握手由接收端率先发起，流程和TCP协议一致。

四次挥手：服务端用户可通过输入命令来停止文件传送，这时将触发挥手操作，流程和TCP协议一致。

RDT发送：使用RDT3.0算法，但加以改进使之**支持不限于0和1的序列号**。因为RDT3.0超时重传在收到非期望数据包时不需要停止时钟，所以在设置接收超时的基础上，使用了显式计时和更改超时时间的方法。具体实现见代码部分。

RDT接收：RDT3.0和RDT2.2的接收状态机一致，当收到非期望分组时将重发ACK。

文件分片发送：考虑UDP数据段所能携带的最大大小位1475byte，大于我们的文件分片，在此函数中我们首先将分片拆成大小为NET_BLOCK_LIMIT（1024byte）的分片，仍然为链表结构。之后将每一个分片的二进制内容附在相应的transmission之后，调用RDT发送函数进行发送。

文件分片接收：在接收到分片后，首先根据第一个分片包含的fileInfo结构体初始化文件数据结构内容。之后接收分片形成链表。最后将其分片合并为大小不超过FILE_BLOCK_LIMIT大小的分片的链表。

关键代码展示

文件读取fileLayer.cpp

```
fileBlockHead *fileLayer::loadFile(string filePath) {
    //用于程序内部的文件信息数据结构
    auto filehead = new fileBlockHead();
    //文件分片空间
    auto buff = new char[FILE_BLOCK_LIMIT];
    string fileName = filePath.substr(filePath.find_last_of('/') + 1, -1);
    //fileInfo数据结构
    struct fileInfo info;
    //获取文件大小
    HANDLE handle = CreateFile(filePath.c_str(), FILE_READ_EA,
                               FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
    if (handle != INVALID_HANDLE_VALUE) {
        info.size = GetFileSize(handle, NULL);
        CloseHandle(handle);
    }else{
        info.size = 0;
    }
    //完善fileInfo数据结构
    size_t pos = fileName.find(".");
    string name = fileName.substr(0, pos);
    string type = fileName.substr(pos + 1, -1);
    strcpy(info.name, name.c_str());
    strcpy(info.type, type.c_str());
```

```
    filehead->setFilename(name);
    filehead->setType(type);
    filehead->setSize(info.size);

    ifstream open_file(filePath, std::ios::binary);
    if (!open_file) {
        std::cout << "error" << std::endl;
        return 0;
    }
    //填充数据分片
    bool isFirst = true;
    memcpy(buff, &info, sizeof(fileInfo));
    while (!open_file.eof()) {
        if (isFirst) {
            //对第一个分片特殊处理，加入fileInfo数据结构
            open_file.read(buff + sizeof(fileInfo), FILE_BLOCK_LIMIT -
sizeof(fileInfo));
            isFirst = false;
        } else {
            open_file.read(buff, FILE_BLOCK_LIMIT);
        }
        filehead->addBlock(buff, FILE_BLOCK_LIMIT);
        memset(buff, 0, FILE_BLOCK_LIMIT);
    }
    open_file.close();
    return filehead;
}
```

RDT发送server.cop

```
void server::sendrdt(char *b, unsigned int size) {
    int nRet = 0;
    sockaddr_in &client = clientVec.back();
    int dwSendSize = sizeof(client);
    auto m1 = (transmission *) b;
    //先发送一次，然后进入循环计时等待
    nRet = sendto(sListen, b, size, 0, (SOCKADDR *) &client, sizeof(SOCKADDR));
    curr_seq++;
    if (nRet == SOCKET_ERROR) {
        cout << "sendto Error " << WSAGetLastError() << endl;
        return;
    }
    //设置接收超时
    DWORD timeout = 3000, t1, t2; //3s
    setsockopt(sListen, SOL_SOCKET, SO_RCVTIMEO, (char *) &timeout,
sizeof(timeout));
    //尝试接收数据
    char buffertemp[4096];
    auto p = (transmission *) buffertemp;
    t1 = timeGetTime();
    nRet = recvfrom(sListen, buffertemp, 4096, 0, (SOCKADDR *) &client,
&dwSendSize);
    // 等待ack
    //此时要么是收到了东西、要么是超时了
```

```
    while (1) {
        // 接收出错
        if (nRet == SOCKET_ERROR) {
            // 确实是接收超时
            if (WSAGetLastError() == 10060) {
                cout << "超时了" << endl;
                cout << "send2 seq ok_seq" << m1->seq << " " << m1->ok_seq <<
endl;
                //再次发送数据
                nRet = sendto(sListen, b, size, 0, (SOCKADDR *) &client,
sizeof(SOCKADDR));
                if (nRet == SOCKET_ERROR) {
                    cout << "sendto Error " << WSAGetLastError() << endl;
                    return;
                }
                cout << "超时返回初值" << endl;
                timeout = 3000;
                setsockopt(sListen, SOL_SOCKET, SO_RCVTIMEO, (char *) &timeout,
sizeof(timeout));
                //再次等待
                t1 = timeGetTime();
                nRet = recvfrom(sListen, buffertemp, 4096, 0, (SOCKADDR *)
&client, &dwSendSize);
                cout << "rcev2 seq ok_seq" << p->seq << " " << p->ok_seq <<
endl;
                // 返回等待ack状态
                continue;
            }
            else {
                // 超时以外的未知错误
                cout << "recv Error " << WSAGetLastError() << endl;
                return;
            }
        } else {
            //正常收到
            if (p->cond == ack && p->ok_seq == curr_seq &&
checksumchecker(buffertemp, p->seq - client_seq)) {
            //发送成功返回
                break;
            } else {
                cout << "损坏或不符合预期" << endl;
                //丢弃无用数据包
                memset(buffertemp, 0, 4096);
                t2 = timeGetTime();
                cout << "重置超时 " << 3000 - (double) (t2 - t1) << " ms" << endl;
                timeout = 3000 - (t2 - t1);
                setsockopt(sListen, SOL_SOCKET, SO_RCVTIMEO, (char *) &timeout,
sizeof(timeout));
                //再次等待
                t1 = timeGetTime();
                //发送重复ACK
                nRet = recvfrom(sListen, buffertemp, 4096, 0, (SOCKADDR *)
&client, &dwSendSize);
                cout << "rcev3 seq ok_seq" << p->seq << " " << p->ok_seq <<
endl;
```

```
                continue;
            }
        }
    }
}
```

遇到的问题

曾经在接受时重复扣去transmission的大小，导致文件中隔一段就有数个字节为全0，修改后传输正常。