实验3-1 实验报告

学号：1911396

姓名：曾泉胜

本次实验我直接完成了回退N步（GBN）的滑动窗口协议。**当其窗口大小为1，序号空间为[0,1]时，就是一个停等协议。**使用该协议，完成了单向的文件传输。

协议设计

协议结构如下所示，总长度为10240字节，显然，在后续的实验中，还要添加更多字段。

```
#define DATA_SIZE 10231
#define ACK_FLAG 0x1
#define FIN_FLAG 0x2
#define SYN_FLAG 0x4

#pragma pack(1)
struct rdt_t
{
    uint16_t sum = 0;           // 校验和
    uint8_t flag = 0;           // 标志位
    uint32_t seqnum = 0;        // 序列号
    uint16_t dataLen = 0;       // 数据长度
    uint8_t data[DATA_SIZE] = {0}; // 数据
};
#pragma pack()
```

打包与解包

在发送前打包数据时计算校验和：每2字节读取打包后的协议包，加到校验和字段，溢出则回卷。

```
void make_pkt(rdt_t* pktBuf, uint8_t flag, uint32_t seqnum, uint8_t* data,
uint16_t dataLen){
    assert(dataLen <= DATA_SIZE);
    pktBuf->sum = 0;
    pktBuf->flag = flag;
    pktBuf->seqnum = seqnum;
    pktBuf->dataLen = dataLen;
    memcpy_s(pktBuf->data, DATA_SIZE, data, dataLen);
    uint16_t validLen = sizeof(rdt_t) - DATA_SIZE + dataLen;
    uint16_t i = 0;
    uint16_t* p = (uint16_t*)pktBuf;
    uint32_t sum = 0;
    for(; i < validLen; i += 2){
        sum += *p;
        rollback(sum);
        p++;
    }
    if (i > validLen) {
```

```
        p--;
        sum += *(uint8_t*)p;
        rollback(sum);
    }
    pktBuf->sum = ~(sum & 0xFFFF);
}
```

在接收后，使用相同的算法，计算出校验和并与校验和字段比较。该算法可以发现一位错，一位错也是大概率事件。

```
bool not_corrupt(rdt_t* pktBuf){
    uint32_t sum = 0;
    uint16_t i = 0;
    uint16_t* p = (uint16_t*)pktBuf;
    uint16_t validLen = sizeof(rdt_t) - DATA_SIZE + pktBuf->dataLen;
    for(; i < validLen; i += 2){
        sum += *p;
        rollback(sum);
        p++;
    }
    if (i > validLen) {
        p--;
        sum += *(uint8_t*)p;
        rollback(sum);
    }
    return sum == 0xFFFF;
}
```

符号/名词解释

符号	含义
`base`	滑动窗口中第一个已发送待确认数据包的序列号，初始为0
`nextseqnum`	滑动窗口中最后一个已发送待确认数据包的序列号，初始为0
`N`	滑动窗口大小
`NUM_SEQNUM`	序列号空间为 `[0, NUM_SEQNUM-1]`，至少为 `2N`
`TIMEOUT`	发送方超时重发时间
`INTEVAL`	检查定时器的时间间隔
`expectedseqnum`	接收方期望的序列号，初始为0
`sendto`	winsock提供的udp发送函数，不可靠
`rdt_send`	自己写的可靠发送函数

程序设计

我的GBN发送方和接收方在建立连接后与断开连接前与教材上的描述没有什么差别，不过在数据结构，事件响应上，我的设计和实现方案如下：

发送方

发送方有3个需要相应的事件，理论上描述如下：

事件	处理线程	动作
上层调用 `rdt_send`	主线程	如果滑动窗口满了，阻塞；如果滑动窗口未满，封装序号为nextseqnum的数据包并放入滑动窗口的相应位置（注意该位置可能由于滑动窗口没有前推而导致已经存在了一个数据包），在放入数据包后使得滑动窗口满了，则启动计时器。
收到一个包	接收线程	如果收到的确认序号不是base的下一个位置，什么也不做；只有收到的确认序号是base的下一个时，才“前推”滑动窗口，之后如果没有待确认的数据包，则停止计时，否则重启定时器。
计时器启动，且到达超时时间	计时线程	重新发送所有已发送但是未确认的数据包。

在实现中，一些细节的处理并没有那么轻松：

滑动窗口的大小是固定的，但是需要在数据包序列上滑动，于是我使用**数组存储的循环队列**保存已发送但是未确认的数据包，这个循环队列的容量是滑动窗口的大小N，使用数组存储是为了满足收到冗余ACK时，能够随机访问之前发过的一个包。滑动窗口的“前推”事实上是由两个步骤完成的：在接收线程中，收到确认序号为 `(base+1)%(NUM_SEQNUM)` 时， `base` 才会更新为该值，同时循环队列将队首出队，滑动窗口“左消”；在主线程中，序号为 `nextseqnum` 的数据包应当放在循环队列的 `(nextseqnum-base+NUM_SEQNUM)%NUM_SEQNUM` 时，只有当该值为循环队列的大小时，才会入队一个新的数据包，将 `nextseqnum` 更新为 `(nextseqnum + 1) % NUM_SEQNUM` ，滑动窗口“右推”。

此外，序号空间的大小至少为滑动窗口大小的两倍，不能简单使用循环队列的大小来得到发送方下次发送的序列号，需要单独维护 `base` 和 `nextseqnum` 在序号空间中此消彼长。 `base` 到 `nextseqnum` 的距离总是等于循环队列的大小，在 `DEBUG` 模式下，我的程序测试了这一点，说明各个线程对循环队列的访问是串行化的，没有出现不同步的问题。

定时器本质上就是记录了开始时间，经过一个 `INTEVAL` 的间隔检查一下距离开始时间有多久，超过 `TIMEOUT` 则超时， 由于主线程和接收线程都会阻塞，因此计时器需要一个单独的“守护”线程。

接收方

GBN的接收方只有一个任务：收到数据包，校验并将其序列号与自己期望的序列号 `expectedseqnum` 比对，如果相同，则接收该数据包， `expectedseqnum` 更新为 `(expectedseqnum + 1) % NUM_SEQNUM` ；否则，抛弃该数据包。最后向发送方发回序号为 `expectedseqnum` 的ACK。主线程中就可以搞定。

连接与断开连接

由于是单向传输，即**接收端不需要保证自己发出的数据包发送端一定会收到**，我没有把连接与断开连接的过程做得那么复杂。我在接收端统计文件传输时间，因此我只保证无论谁先启动，接收端能够**确切**地知道什么时候，开始了文件中第一个字节的传输，什么时候完成了文件的传输。

连接

发送端启动接收线程但主线程中开始发送数据之前，先不断发送（`sendto`）带有SYN标记的数据包，直到接收线程收到了接收方返回的带有SYN ACK标记的数据包。之后就不必要回复ACK而直接开始发送数据就可以了，因为接收方不在乎它发的数据包能不能可靠到达发送方。

断开连接

在发送方的主线程读完文件时，使用`rdt_send`发出FIN包，与SYN包不同的是，FIN包带有序列号，即当接收方接收该FIN包时，前面的数据包一定都接收到了，接收方停止计时并在结束之前向发送方`sendto`一个FIN ACK（不在乎发送方收没收到）。发送方收到FIN ACK后，结束接收线程，结束主线程，没收到就手动结束。

测试

正确性

关于能否正确地传输图片不便于在报告中展示，已在课堂上给助教展示。

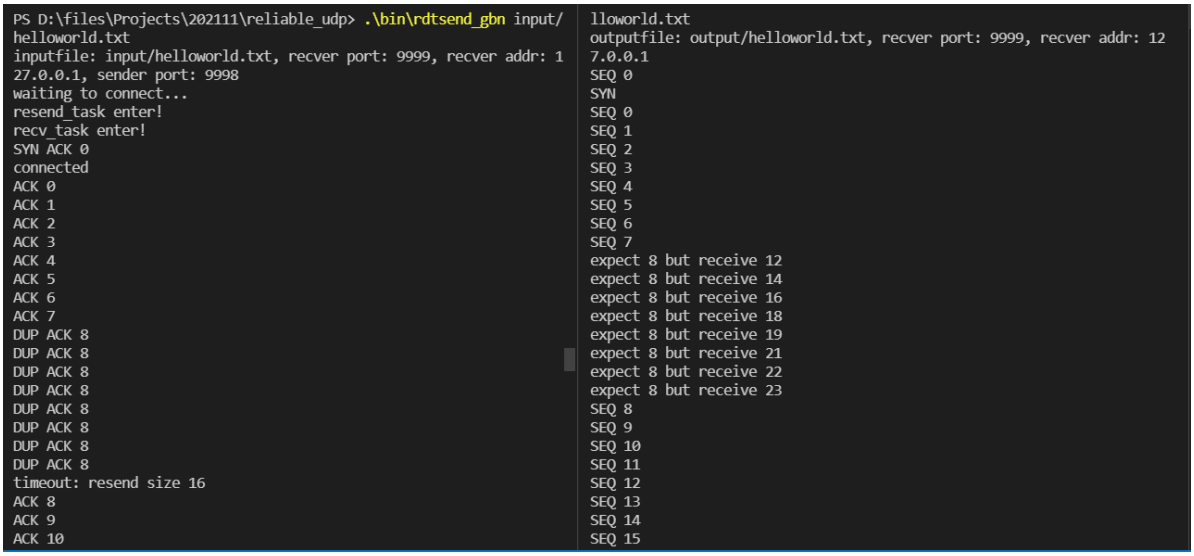
性能

目前程序中可以调整的参数有：`TIMEOUT`,`INTEVAL`,`N`,`NUM_SEQNUM`以及数据包大小。更详细的对比实验放在3-4中进行，下面仅展示在本地回环上，改变窗口大小的测试结果。

使用大小11968994 Bytes的文件，超时时间为50ms，序列号数目为窗口大小2倍：

窗口大小N	传输时间（ms）	吞吐率（Mb/s）
1	164.15	583.31
2	123.60	774.71
4	109.33	875.77
8	103.03	929.40
16	65581.88	1.5

当窗口大小达到16时，发现接收方收到了大量的失序数据包，接收方因而收到了大量的DUP ACK，目前没有快速重传机制，只能等待超时重传所有的包。



在后续的实验中应当探索原因并改进。