实验3-3 基于UDP服务设计可靠传输协议并编程实现

一、实验要求

实验3-1，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。

实验3-2，在实验3-1的基础上，将停等机制改成基于滑动窗口的流量控制机制，采用固定窗口大小，支持累积确认，完成给定测试文件的传输。

实验3-3，在实验3-2的基础上，选择实现一种拥塞控制算法，也可以是改进的算法，完成给定测试文件的传输。

二、报告说明

本次实验报告会讲报文格式，交互流程，滑动窗口，重点在慢启动，拥塞避免，快速重传，快速恢复。

之前的报告中已经详细说明了握手过程，挥手过程，差错重传和累计确认，指定文件写入缓冲区的内容，所以这里不再说明。

三、报文格式

本次实验在上次的基础上，适当更改了发送序号seq的宽度，将4字节的seq变成2字节seq和2字节ack，但报文长度仍然不变，报文头也不变，报文共0x4000字节

<table>
<tr><td></td><td>0</td><td>4</td><td>8</td><td>12</td><td>16</td><td>20</td><td>24</td><td>28</td></tr>
<tr><td>0</td><td colspan="2">ACK</td><td colspan="2">SYN</td><td colspan="2">FIN</td><td>SF</td><td>EF</td></tr>
<tr><td>4</td><td colspan="8">源IP</td></tr>
<tr><td>8</td><td colspan="8">目的IP</td></tr>
<tr><td>12</td><td colspan="4">源端口</td><td colspan="4">目的端口</td></tr>
<tr><td>16</td><td colspan="4">发送序号seq</td><td colspan="4">确认序号ack</td></tr>
<tr><td>20</td><td colspan="8">报文长度filelength</td></tr>
<tr><td>24</td><td colspan="4">index</td><td colspan="4">校验和checksum</td></tr>
<tr><td>28</td><td colspan="8">⋮</td></tr>
<tr><td>⋮</td><td colspan="8">⋮</td></tr>
<tr><td>⋮</td><td colspan="8">⋮</td></tr>
<tr><td>⋮</td><td colspan="8">⋮</td></tr>
</table>

```
#define SEND_MAX 0x4000
#define SEND_TOP 0x1C
char sendBuf[SEND_MAX];
char recvBuf[SEND_MAX];
//五个flag标志位
sendBuf[0]=1;//ACK确认字符
sendBuf[1]=1;//SYN建立连接
sendBuf[2]=1;//FIN断开连接
sendBuf[3]=0x10//SF开始发送文件
sendBuf[3]=0x1//EF结束发送文件
```

四、交互流程

本次实验选择实现GBN，即实现重点为累计确认，超时重传，滑动窗口等。

首先client发送SYN报文和server建立连接，server回复ACK报文确认连接。

开始出传输文件，正常情况下seq随着报文个数升高而升高，server回复ACK，并且发送的ack=收到相应的seq，表示已经收到，窗口根据ack按照相应的大小向前滑动，其中窗口大小根据慢启动拥塞避免等过程中的变化而变化，下面会详细介绍。使用队列记录每个已经发出并且没有收到回复报文ack的发送，判断队列中第一个记录时间是否超时，超时则清空队列，并从第一个开始重传，这是快速重传。同时判断client端是否接收到同一个ack报文超过三次，超过三次同样清空队列，快速恢复。下面会具体介绍快速重传和快速恢复。

最后，client发送FIN报文表示结束传输，server回复ACK报文，二者成功断开连接。

五、慢启动

如果立即将大量的数据注入到网络中可能会出现网络的拥塞。先探测一下网络的状况，如果网络状况良好，发送方每发送一次文段都能正确的接受确认报文段，那么就从小到大的增加拥塞窗口的大小，即增加发送窗口的大小。

首先拥塞窗口cwnd初始设为1，在发送seq=1并且接收到ack=1后，将cwnd增大为2，发送seq=2,seq=3，接受到确认后cwnd增加为4，每次发送成功确认都加倍窗口cwnd。当cwnd达到慢启动峰值线ssthresh时，进入拥塞避免阶段。

六、拥塞避免

为了防止cwnd增加过快而导致网络拥塞，所以需要设置一个慢开始门限ssthresh状态变量，让超过一定大小的窗口缓慢增加，比如窗口为10，每次窗口为10，每次加一，按照线性增长，而不是像上面所说的那样按指数增长。

> cwnd < ssthresh　　慢启动算法（cwnd*=2）
>
> cwnd >= ssthresh　　拥塞控制算法（cwnd++）

七、快速重传

当窗口中的第一个包已经超时，立即重传，与GBN中实现的超时重传一样。但与上一次不同的是这里不是固定窗口大小，因为超时，所以认定网络出现异常，不稳定，需要将ssthresh变为cwnd的一半，cwnd变为1

> 超时　　快速重传（ssthresh=cwnd/2　cwnd=1）

八、快速恢复

当连续收到三个相同的ack时，立即重传。但窗口变化不像快速重传那样，因为还能收到ack说明网络连接可能没问题，所以不需要将cwnd变为1，让重传代价太大。

> 连续收到3个重复ack　快速恢复（ssthresh=cwnd/2　cwnd=ssthresh+3）

九、发送文件报文、滑动窗口变化

发送文件报文，在发送报文前设置计时器，并记录下结束后的时间。

本次实验采用多线程接收发送，主线程为发送，子线程为接收。使用队列记录窗口，内容包括<报文序号，发送时间>。快速重传，慢启动，拥塞避免，快速恢复在这里有所体现。

主线程：

循环判断

- 判断是否成功完的个数，发完退出
- 判断窗口是否已满，是否已经发送全部的报文，没满并且没发送全部的则继续发送，并且在窗口队列里加入这个seq和发送时间，如果满了或者已经发送了所有的，就直接退出运行后面的内容或者运行子线程
- **快速重传**，判断发送时间是否超过规定时间，窗口清空，并且设置ssthresh = cwnd / 2，cwnd = 1，设置窗口base为原来窗口的第一个报文。

子线程：

判断是否接收到消息，如果接收到消息

- 判断ack是否在窗口内，如果大于等于窗口最前端，窗口移动到ack+1。将上一次接收到的报文lastack设为收到报文的seq，将ack重复接收计数lastacksum重置为1。
  - 如果cwnd<ssthresh，采取**慢启动**，cwnd *= 2
  - 如果cwnd >= ssthresh，采取**拥塞避免**，cwnd++
- 如果ack不再窗口内
  - 判断重复接收计数lastacksum是否为3
    - 是，则采用**快速恢复**机制，ssthresh = cwnd / 2，cwnd = ssthresh + 3，发送ack+1报文，重置lastacksum为1.
    - 不是，则判断收到的ack是否为上一次收到过的，是则lastacksum++

每次接收到消息时输出窗口变化。

client端：

主线程判断发送

```
int t_start = clock();
while (1)
{
    if (recvBuf[2] == 1)
        break;
    //判断是否成功发送所有报文
    WaitForSingleObject(hMutex, INFINITE);
    if (base >= seq + sum + 1)
    {
        cout << "全部发送" << endl;
        break;
    }
    ReleaseMutex(hMutex);
    //判断窗口是否满
    WaitForSingleObject(hMutex, INFINITE);
    while (1)
    {
        if (((has_send - base) < cwnd) && has_send < seq + sum + 1)
        {
            want_to_sendfile(++has_send);
            sendto(sockSrv,sendBuf,sizeof(sendBuf),0,
(SOCKADDR*)&addrServer,len);
            timer_list.push(make_pair(has_send, clock()));
        }
        else break;
    }
```

```
    ReleaseMutex(hMutex);
    WaitForSingleObject(hMutex, INFINITE);
    if (timer_list.size() != 0)
    {
        if ((clock() - timer_list.front().second) > TIMEOUT)
        {
            //快速重传
            has_send = base - 1;
            ssthresh = cwnd / 2;
            cwnd = 1;
            while (timer_list.size()) timer_list.pop();
        }
    }
    ReleaseMutex(hMutex);
}
int t_end = clock();
```

子线程判断接收

```
HANDLE hMutex = NULL;//互斥量
DWORD WINAPI recvthread(LPVOID lpParamter)
{
    SOCKET sockSrv = (SOCKET)(LPVOID)lpParamter;
    while (1)
    {
        if (recvfrom(sockSrv, recvBuf, sizeof(recvBuf), 0, (SOCKADDR*)&addrServer, &len) != -1  && recvBuf[0]   && verifychecksum())
        {
            WaitForSingleObject(hMutex, INFINITE);
            int ack = (((unsigned int)((unsigned char)recvBuf[18]) << 8)) + (((unsigned int)(unsigned char)recvBuf[19]));

            if (ack >= base && ack <= has_send)
            {
                while (timer_list.size() != 0 && (timer_list.front().first <= ack))
                {
                    timer_list.pop();
                    base++;
                }
                if (cwnd < ssthresh) cwnd *= 2;//慢启动
                if (cwnd >= ssthresh) cwnd++;//拥塞避免
                lastack = base - 1;
                lastacksum = 1;
            }
            else
            {
                if (lastacksum >= 3)
                {
                    //快速恢复，设置cwnd，ssthresh，发送相应报文
                    ssthresh = cwnd / 2;
                    cwnd = ssthresh + 3;
                    base = has_send = ack + 1;
                    want_to_sendfile(ack+1);
                    sendto(sockSrv, sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrServer, len);
                    while (!timer_list.empty())timer_list.pop();
```

```
                    timer_list.push(make_pair(has_send, clock()));
                    lastacksum = 1;
                }
                else
                {
                    if (ack == lastack)
                    {
                        //收到重复的ack了
                        lastacksum++;
                    }
                }
            }
            cout<<"窗口大小 "<<has_send-base << "  左 " << base << "  右 " << has_send << endl;
            ReleaseMutex(hMutex);
        }//释放互斥量锁
    }
    return 0L;//表示返回的是long型的0
}
```

server端：

服务端如果接收到文件报文，判断校验和

- 如果校验和正确
  - 计算本次报文的序列号，如果序列号比上次大1，说明接收正确
    - 如果该报文为文件报文开始，即recvBuf[3] == 0x10，将名字写入name，设置ack发送报文。
    - 如果该报文为文件报文结束，即recvBuf[3] == 0x01，将内容写入content后，设置ack发送报文，结束循环。
    - 如果该报文为文件报文内容，即recvBuf[3] == 0x00，根据index，将内容写入相应的content，设置ack发送报文。
  - 如果序列号比上次的小或者等于上次接收的，则发送对应报文，但不做任何读报文操作
  - 如果不是以上情况，重复发上次发送过的报文，让client端窗口不能滑动，直到超时来快速重传或者重复三次ack来快速恢复。
- 如果校验和不正确，直接发送ACK=0

```
while (1)
{
    if (recvfrom(sockSrv,recvBuf,sizeof(recvBuf),0,(SOCKADDR*)&addrClient,&len)!=-1)
    {
        if (recvBuf[2] == 1)break;
        thisseq = (((unsigned int)((unsigned char)recvBuf[16]) << 8))
            + (((unsigned int)(unsigned char)recvBuf[17]));
        if (verifychecksum())
        {
            if (recvseq == thisseq - 1)
            {//如果上次收到的等于这次收到-1，则按顺序接收
                //如果上次收到的大于这次收到的-1，则回复响应的
                if (recvBuf[3] == 0x10)
                {
                    setname();
                }
                if (recvBuf[3] == 0x00)
```

```
                {
                    setcontent(calculateindex());
                }
                if (recvBuf[3] == 0x01)
                {
                    size = calculateindex();
                    setcontentend(size);
                }
                sendBuf[0] = 1;//ACK=1
                getseqsetack();
                calculatechecksum();
                recvseq = thisseq;
                sendto(sockSrv,sendBuf,sizeof(sendBuf),0,
(SOCKADDR*)&addrClient,len);
            }
            else
            {
                if (recvseq > thisseq - 1)
                {
                    sendBuf[0] = 1;//ACK=1
                    getseqsetack();
                    calculatechecksum();
                    sendto(sockSrv,sendBuf,sizeof(sendBuf),0,
(SOCKADDR*)&addrClient, len);
                }
                else
                {
                    sendto(sockSrv,sendBuf,sizeof(sendBuf),0,
(SOCKADDR*)&addrClient, len);
                }
            }
        }
        else
        {
            sendBuf[0] = 0;//ack=0
            calculatechecksum();
            sendto(sockSrv,sendBuf,sizeof(sendBuf),0,
(SOCKADDR*)&addrClient,len);
        }
    }
}
```

setcontent():

```
void setcontent(int index)
{
    for (int i = 0; i < SEND_MAX-SEND_TOP; i++)
    {
        content[index - 1][i] = recvBuf[i + 28];
    }
}
```

setcontentend():

```
void setcontentend(int index)
{
    endfilesize = (((unsigned int)((unsigned char)recvBuf[20]) << 24))
        + (((unsigned int)((unsigned char)recvBuf[21]) << 16))
        + (((unsigned int)((unsigned char)recvBuf[22]) << 8))
        + (((unsigned int)(unsigned char)recvBuf[23]));
    for (int i = 0; i < endfilesize; i++)
    {
        content[index - 1][i] = recvBuf[i + 28];
    }
}
```

计算平均吞吐率

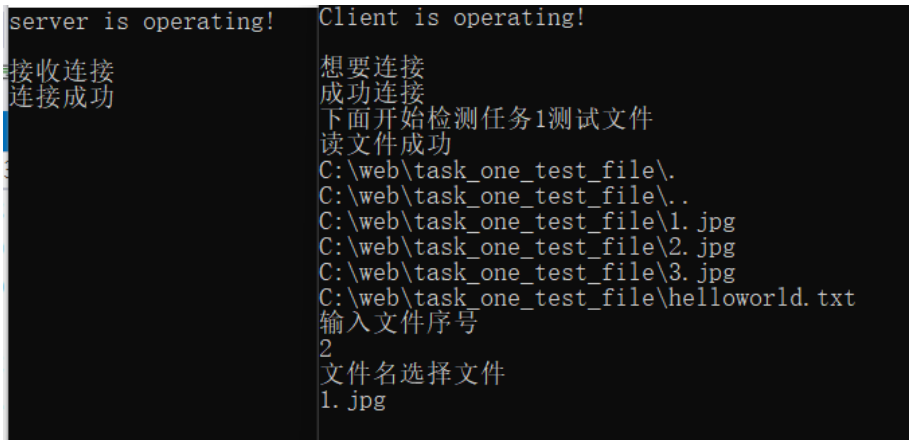
根据计算得到的文件文件报文个数计算平均吞吐率。

```
cout << "发送" << (sum * 16356 + endfilesize) << "字节" << (t_end - t_start)
<< "毫秒" << endl;
cout << "平均吞吐率" << (sum * 16356 + endfilesize) * 8 * 1.0 / (t_end -
t_start) * CLOCKS_PER_SEC << " bps" << endl;
```

执行界面输出日志

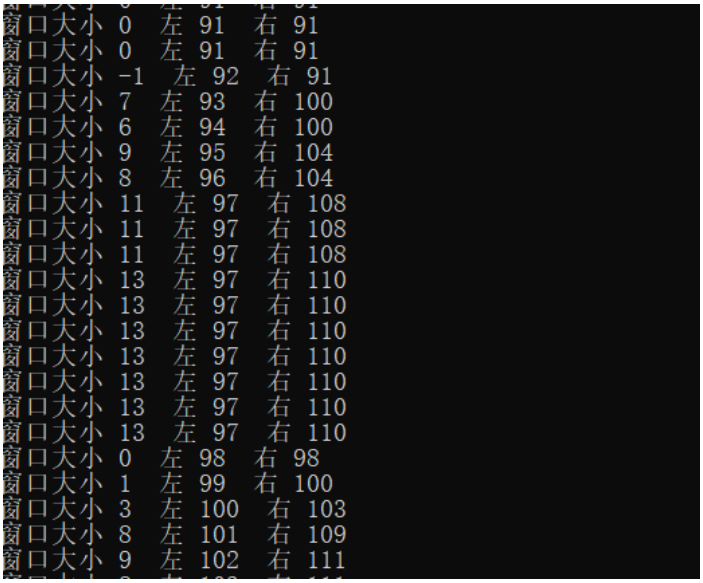
丢包率0%

建立连接

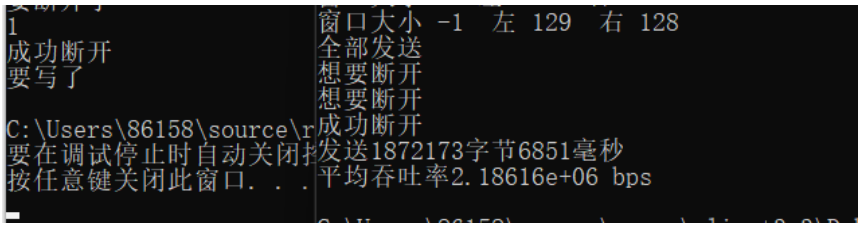


发送文件

client端



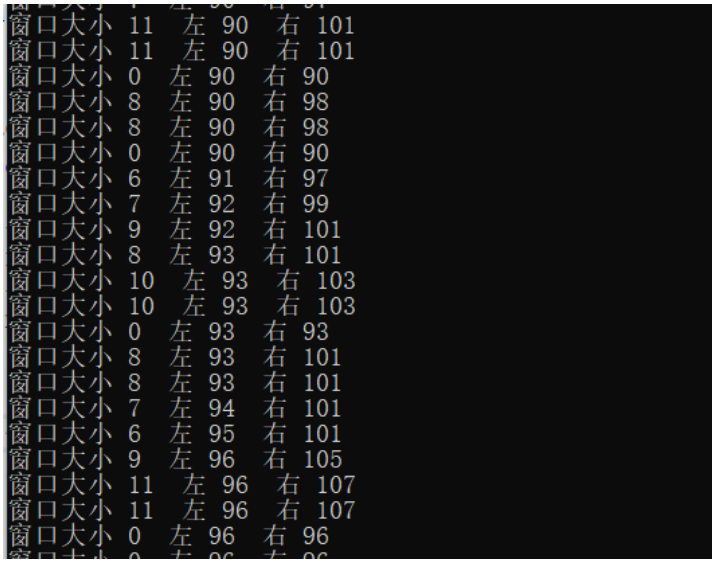
断开连接及吞吐率



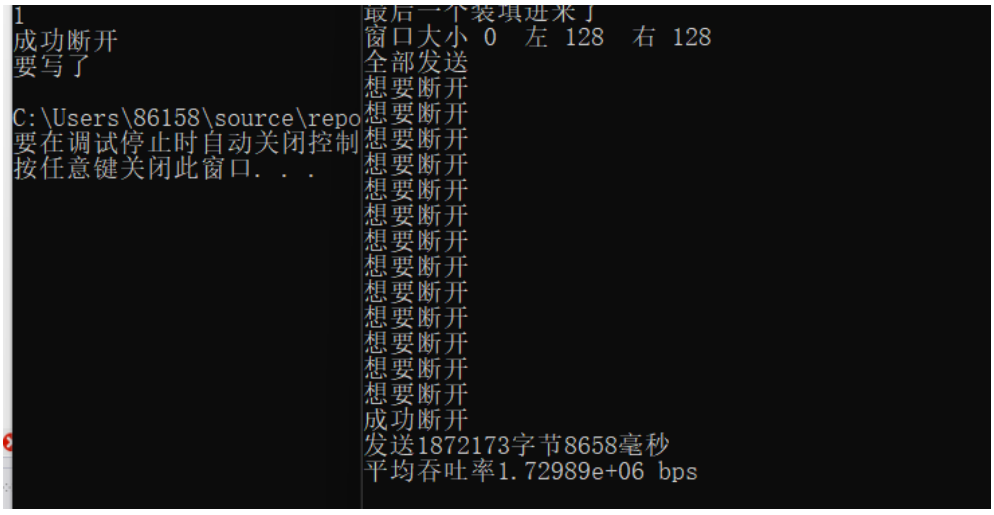
丢包率10%

发送文件

client端



断开连接及吞吐率



延时为10ms

建立连接

```
server is operating!
接收连接
连接成功
```

```
Client is operating!
想要连接
想要连接
想要连接
想要连接
想要连接
想要连接
想要连接
成功连接
下面开始检测任务1测试文件
读文件成功
C:\web\task_one_test_file\.
C:\web\task_one_test_file\..
C:\web\task_one_test_file\1.jpg
C:\web\task_one_test_file\2.jpg
C:\web\task_one_test_file\3.jpg
C:\web\task_one_test_file\helloworld.txt
输入文件序号
```

发送文件

client端

```
窗口大小 0  左 82  右 82
窗口大小 10  左 82  右 92
窗口大小 10  左 82  右 92
窗口大小 8  左 84  右 92
窗口大小 10  左 85  右 95
窗口大小 9  左 86  右 95
窗口大小 12  左 87  右 99
窗口大小 11  左 88  右 99
窗口大小 15  左 88  右 103
窗口大小 15  左 88  右 103
窗口大小 0  左 88  右 88
窗口大小 10  左 88  右 98
窗口大小 10  左 88  右 98
窗口大小 8  左 90  右 98
窗口大小 10  左 91  右 101
窗口大小 9  左 92  右 101
窗口大小 8  左 93  右 101
窗口大小 13  左 94  右 107
窗口大小 15  左 94  右 109
窗口大小 15  左 94  右 109
```

断开连接及吞吐率

Microsoft Visual St

```
server is operating
接收连接
连接成功
要断开了
1
成功断开
要写了

C:\Users\86158\sour
要在调试停止时自动
按任意键关闭此窗口.
```

Microsoft Visual Studio 调试控制台

```
窗口大小 -1  左 126  右 125
最后一个装填进来了
窗口大小 1  左 127  右 128
窗口大小 0  左 128  右 128
窗口大小 -1  左 129  右 128
全部发送
想要断开
想要断开
想要断开
想要断开
想要断开
想要断开
想要断开
想要断开
想要断开
想要断开
想要断开
成功断开
发送1872173字节10081毫秒
平均吞吐率1.4857e+06 bps
```