



哈尔滨工业大学
Harbin Institute of Technology

计算机网络 课程实验报告

实验名称	可靠数据传输协议——停等、GBN、SR 协议的实现					
姓名	王炳轩		院系	信息安全		
班级	2003021		学号	120L022115		
任课教师	刘亚维		指导教师	刘亚维		
实验地点	格物 213		实验时间	2022 年 10 月 14 日		
实验课表现	出勤、表现得分(10)		实验报告 得分(40)		实验总分	
	操作结果得分(50)					
教师评语						

实验目的：

本次实验的主要目的。

理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

实验内容：

概述本次实验的主要内容，包含的实验项等。

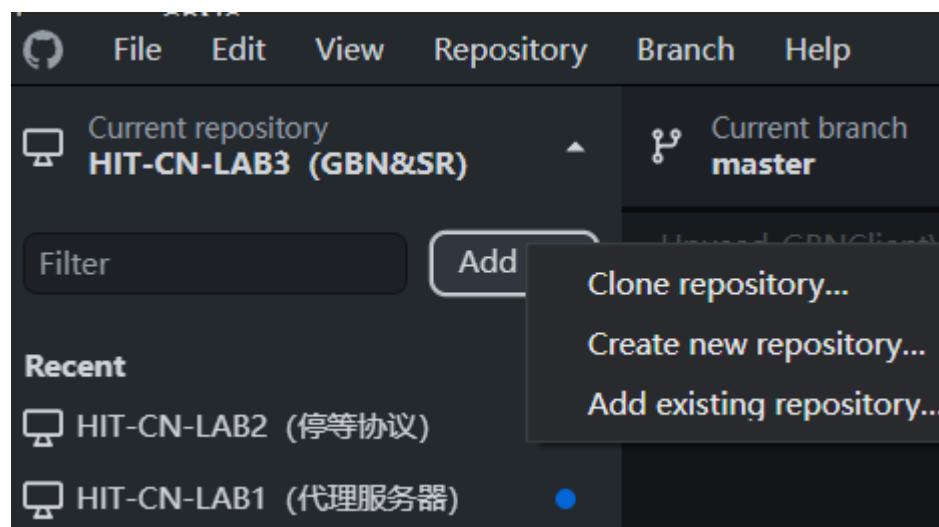
- 1) 基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。
- 2) 模拟引入数据包的丢失，验证所设计协议的有效性。
- 3) 改进所设计的停等协议，支持双向数据传输；（选作内容，加分项目，可以当堂完成或课下完成）
- 4) 基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。（选作内容，加分项目，可以当堂完成或课下完成）
- 5) 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。
- 6) 模拟引入数据包的丢失，验证所设计协议的有效性。
- 7) 改进所设计的 GBN 协议，支持双向数据传输；（选作内容，加分项目，可以当堂完成或课下完成）
- 8) 将所设计的 GBN 协议改进为 SR 协议。（选作内容，加分项目，可以当堂完成或课下完成）

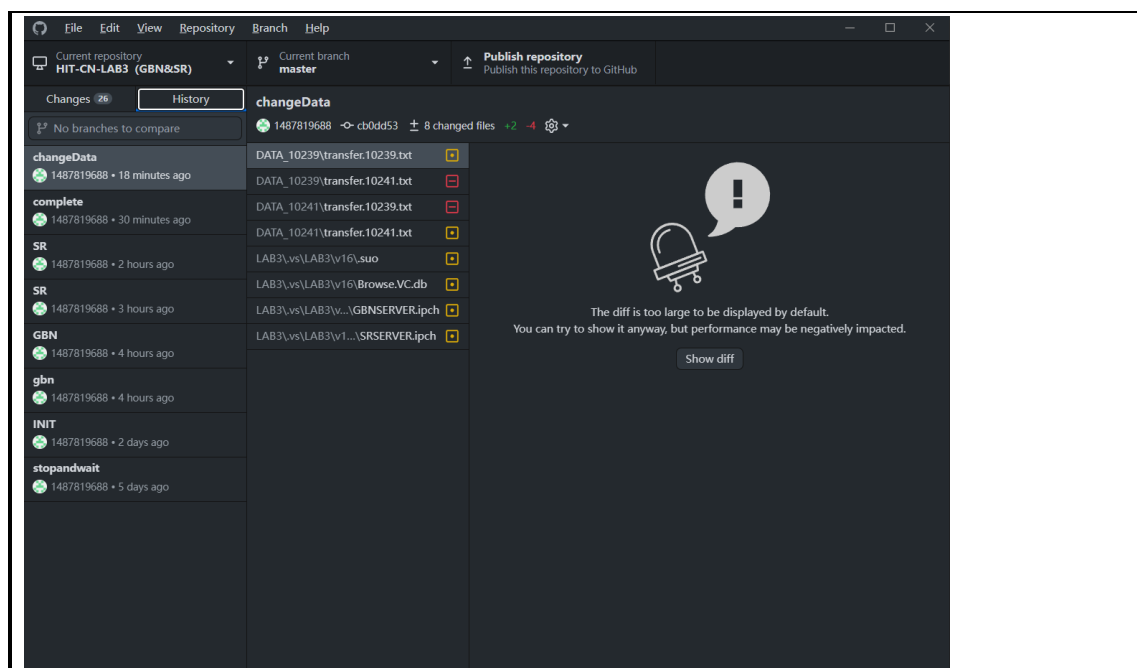
实验过程：

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

一、创建Git本地仓库，用以保存实验文件的历史版本

下载安装Github Desktop应用，点击当前仓库列表（Current Repository），点击新建（Add）- 本地仓库（Create new repository）。输入仓库名为LAB2、3，点击确认创建。

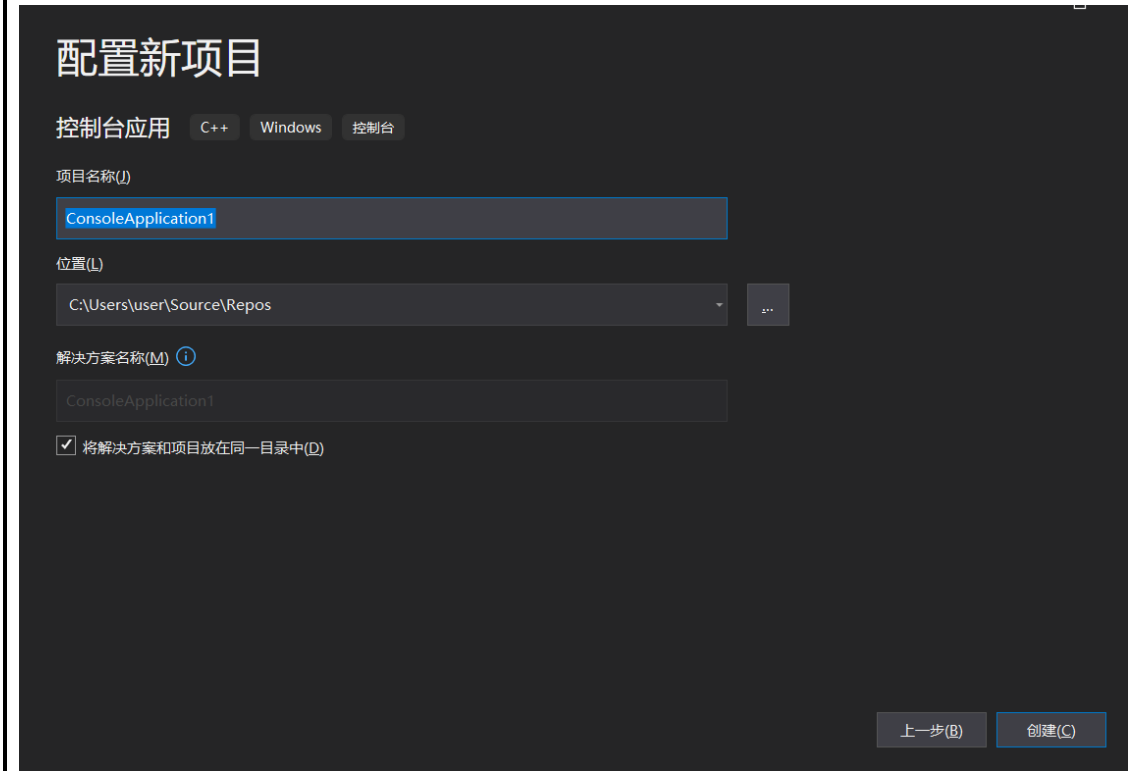
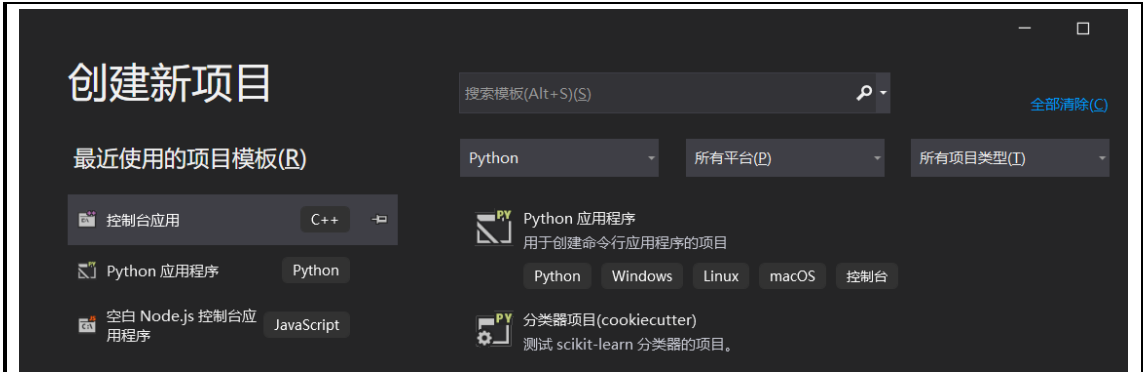




二、配置Visual Studio 2019，新建解决方案LAB2、3。

打开VS2019，选择创建新项目，选择C/C++控制台程序，输入项目名称和存储位置，点击创建。





三、构建实验框架：终端基本交互控制台（实现文件传输应用）

数据传输均使用UDP实现，自定义了一种连接方式（两次握手）。在一次连接中，可以双向传输多个文件，但文件传输采用半双工模式，即：在某个文件传输的这段时间内只能有一端发送文件（但双端可以发送控制报文），等一端传输一个文件完成后，另一端可以开始发送或继续接收下一个文件。

使用<stdio.h>在main函数中循环读取字符串，用户可以输入“S”（以默认方式设置保存位置并启动），“s%s”（指定保存位置并启动），“C”（连接默认目标），“c%s”（连接指定目标），“F”（发送默认测试文件），“f%s”（发送指定文件），“R”（开始接收文件），“Q或q”退出。

默认的保存位置在“../Data_端口号/”

默认的发送文件在“保存位置/transfer.端口号.txt”

默认连接目标是127.0.0.1。

使用宏定义来存储这些信息。

```
#define DST_IP "127.0.0.1"
#define TRANS_FILE_DEFAULT "transfer"
#define FILE_NAME_FORMAT "%s/%s.%d.txt"
```

```
#define FILE_DIR_DEFAULT "../DATA/"
```

为了测试方便，使用宏定义设置自动启动并连接，并启动另一个进程。

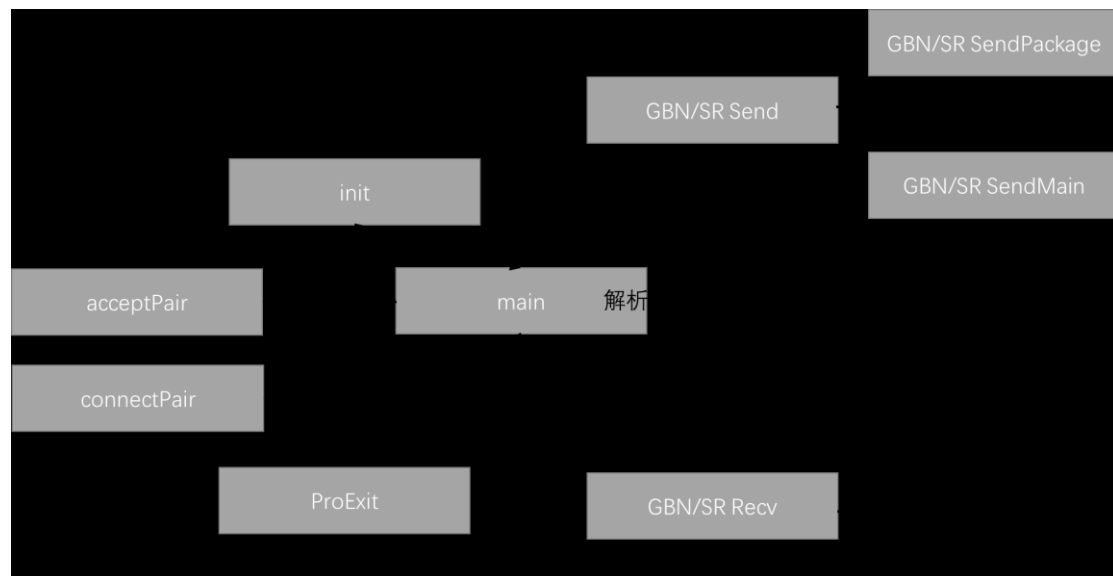
```
#define AUTO_START true
```

当程序启动时，读取命令行判断是不是子进程，设置好进程套接字端口（主进程的端口-2），然后自动执行“S、C”命令。

例如，用户打开程序后，需要先使用“S、C”命令进行初始化和连接，然后使用“R/F”发送或接收文件。如果文件夹不存在还会自动创建文件夹。

```
scanf_s("%s", cmd, 100);
if (_access(cmd, 0) == -1)
{
    strcpy_s(FILE_DIR, cmd);
    _mkdir(cmd);
}
printf("正在启动传输，文件将保存到文件夹：%s\n", FILE_DIR);

waitForConnect();
Sleep(100);
printf("WBXFileTransfer> ");
```



四、构建实验框架：共同代码头文件ftcommon.h

4.1 初始化相关

定义数据结构，用来存储套接字的相关信息。

```
struct Job {
    SOCKET socket;
    unsigned long srcIp;
    unsigned long dstIp;
    unsigned short srcPort;
    unsigned short dstPort;
```

```

    bool connected;
} job;

```

ftcommon.h是三个协议共同使用的基础函数和数据结构集，包括：基本的初始化函数（init），WSAStartup，获取套接字并保存到job中，以及打开第二个进程，获取文件大小，处理连接请求，封装发送的数据包，实现定时器，获取错误文本，通用退出等。

通用退出如下所示：包含关闭套接字和文件，以及WSACleanup。

```

int proexit(int err) {
    printf("正在关闭客户端! \n");
    if (file != NULL) fclose(file);
    if (job.socketSend != NULL) closesocket(job.socketSend);
    //if (job.socketRecv != NULL) closesocket(job.socketRecv);
    WSACleanup();
    Sleep(3000);
    exit(err);
}

```

创建子进程时，第一个命令行传入“PORT=%d”使用端口号格式化，子进程读入后，端口自动-2，并展示在控制台窗口标题，确保在本机测试不冲突。

```

void createSubProcess(int port, const WCHAR* exefile) {
    STARTUPINFO startupInfo = { 0 };
    startupInfo.cb = sizeof(startupInfo);
    PROCESS_INFORMATION processInfo = { 0 };
    WCHAR cmd[1024] = { 0 };
    swprintf_s(cmd, L"PORT=%d", port);
    if (!CreateProcess(exefile, cmd, 0, 0, true, CREATE_NEW_CONSOLE, 0, 0,
&startupInfo, &processInfo))
    {
        printf("CreateProcess failed\n");
    }
    else
    {
        SUB_PROCESS = true;
        printf("CreateProcess sucssessed\n");
    }
}

```

4.2 数据交换相关

定义以下数据结构用于封装数据包：

```

struct Data {
    char control[4];
    char seq[8];
    char dat[MAXSIZE+2];
};

#define MAXPACK sizeof(Data)
#define MAXSIZE 500 //发送数据报文的最大长度

```

数据包的control表示控制字段，可以指定为以下的值。

```

#define METHOD_SYN_TEXT "SYN" //建立连接
#define METHOD_LEN_TEXT "LEN" //交换文件名和长度
#define METHOD_DATA_TEXT "DAT" //交换数据
#define METHOD_ACK_TEXT "ACK" //接收确认
#define METHOD_FIN_TEXT "FIN" //关闭连接

```

数据包的seq在不同时期表示不同：

在连接请求（SYN）时：表示为端口号。

在文件名交换（LEN）时：表示文件长度。

在交换数据（DAT）和接收确认（ACK）时：表示数据包的序号。

其余时，置0。

数据包的dat字段表示数据内容或文件名，内容长度必须小于等于MAXSIZE。

用户连接使用两次SYN，发送文件使用LEN先传递文件长度和文件名，接收端使用ACK 0确认，使用DAT+seq发送数据，接收端使用ACK+seq确认，接收端在接收到文件长度的数据时发送ACK并结束文件接收，发送端在最后一个ACK接收后结束文件发送。

数据包的序号从1开始，结束于最大值，超过最大序号（分组）个数的文件不进行发送。

注：程序使用了一个整型数组用来存储各个包的状态，因此不能太大。

```

#define MAX_PACKAGE_NUM 100000

```

定义以下函数用以实现数据包的打包和解包：由于发送的数据中不能包含“\0”，因此对每个字段（method、seq）的结尾字符需要做替换“_”，解包时再替换回来。

发送时将数据结构Data*视为char*发送。接收时使用Data*的结构视作char*直接接收。

```

int packData(SOCKET s, const char* method, int seq, char* data, int len)
int unPackData(Data* Buffer, SOCKET s)

```

4.3 定时器

```

#define TIME_OUT 5 //5s

```

以下函数用以实现定时器，首先定义宏TIME_OUT，用以标志超时时间。

需要传入一个指针t来保存和确定上一次的时间，并根据返回值确定是否超时。

```

int timer(time_t* lastt) {
    time_t t = time(NULL);
    if (*lastt == 0) {
        *lastt = t;
        return 0;
    }
    else {
        if (t - *lastt > TIME_OUT) {
            *lastt = 0;
            return 1;
        }
        printf("还剩 %lld s\n", TIME_OUT - t + *lastt+1);
        return 0;
    }
}

```

定义数据结构用于发起的定时器重发线程读取数据（这个数据结构作为定时器的参数，仅SR使用，在GBN中因为需要重发多个数据，已使用新的数据结构PACKAGE定义的全局变量）：

```

struct time {

```

```

SOCKET as;
char* data;
const char* method;
int seq;
//int len;
int* stop;
int readSize;
bool* cancel;
int* targetseq;
time(SOCKET s, char* b, const char* m, int seqq, int* st, int reS, bool* cancell, int
*targets) {
    as = s;
    data = b;
    method = m;
    seq = seqq;
    //len = 1;
    targetseq = targets;
    cancel = cancell;
    stop = st;
    readSize = reS;
}
};
typedef struct time TIME;

```

4.4 用户连接相关

定义以下两个函数和数据用以用户连接：

```

int PASSIVE = true;
int startRecv = false;
void connectPair(int port, int passive)
unsigned int __stdcall acceptPair(void* param)

```

其中PASSIVE表示是否为被动连接（即等待方）。

用户使用上一节构建的交互窗口，使用“S”初始化连接并进入等待状态（启动新线程调用acceptPair）。

必须进入等待状态后，才能使用“C”发起连接，一端用户使用C时，调用connectPair发送连接请求。另一端用户无需使用“C”，因为在acceptPair中，另一端收到连接请求后，是PASSIVE，则自动发起connectPair进行连接确认。双方均通过报文获得对方的IP和端口号。

连接请求和连接确认使用同样的格式（函数），如下所示：

Data { control="SYN",seq=端口号,dat=主机IP }

即A用户使用connectPair后，B用户在acceptPair接收到请求，并启动connectPair进行回连，A用户也在acceptPair接收到请求，但因为不是PASSIVE，不进行回连，自此，连接建立。

五、编写停等协议代码

5.1 接收端

在初始化和连接建立后，用户在交互窗口使用“R”命令后，程序进入接收文件模式，等待连接的另一方发送文件。首先等待LEN控制符，接收后创建由“用户指定的保存位置+LEN

指定的文件名”的文件并获得文件长度，发送ACK 0，循环等待DAT seq，写到文件，发送ACK seq，直到写到完整文件长度。

5.2 发送端

同理，在初始化和连接建立后，用户在交互窗口使用“F”命令后，程序进入发送文件模式，另一方接收文件。首先打开指定的文件，获取长度和文件名，发送LEN控制符，接收到ACK 0，循环读入文件，发送DAT seq，等待ACK seq，直到收到最后一个ACK。

六、编写GBN协议代码

```
#define WINDOW_SIZE 5
```

先使用宏定义窗口大小。

6.1 接收端

在初始化和连接建立后，用户在交互窗口使用“R”命令后，程序进入一个函数gbnrecv，同理先发LEN、等ACK0，然后循环等待DAT seq，遇到乱序的DAT直接丢弃，正确的写到文件，发送ACK seq.....直到写到完整文件长度。

6.2 发送端

发送端较为复杂，需要使用多个线程和定时器。

在初始化和连接建立后，用户在交互窗口使用“F”命令后，程序进入发送主函数gbntansfer，首先打开指定的文件，发送LEN，接收ACK 0，然后使用全局变量设置相关信息后，新建一个线程thread_sendPackage用来循环发送数据，主函数则不断循环接收ACK，并更新数据结构。

以下是两个线程需要共享的全局变量：

```
int Gbase;           //窗口的最小序号
int Gmaxseq;         //窗口最大的序号
int Gseq;            //当前发送的序号
int Gstoparr[MAX_PACKAGE_NUM]; //已完成发送的数据报序号（桶排序，用于停止计时器），本来使用bool，但因SR共用改成int，在SR中用0表示未发送，1表示已正确接收，2（仅接收方）表示已缓存
int GtotalSend;      //已正确接收的发送字节数
int GtotalBlock;     //总数据报数
int GreadSizeArr[MAX_PACKAGE_NUM]; //每个数据报的数据大小
int GFilesize;       //文件总大小
FILE* Gf;            //文件符
SOCKET Gas;          //套接字符
```

在thread_sendPackage中，有while(GtotalSend<GFilesize){while(Gseq<Gbase)}字段，循环读取文件、发送数据，Gseq/Gbase与主线程共享，主线程收到ACK时修改，这边就能不断发送。每次发送数据时，都会启动一个与seq关联的定时器，定时器也需要共享字段，如下所示：

```
typedef struct Package {
    int stoparr = 0;           //是否已收到ACK，并关闭计时器
    bool cancelarr = 0;        //是否因为收到更前的错误ACK，将统一发送新的（从更前到最新seq），需要取消当前计时器
    const char* method = NULL; //发送数据的控制字段类型
    int targetseq = 0;          //统一发送新的时，最新的seq
    int size = 0;               //发送数据的内容大小
```

```
char* FILEBUFFER =0;           //发送数据的内容
}PACKAGE;
```

```
PACKAGE PACK[MAX_PACKAGE_NUM];
```

每次发送数据时，需要在更新此全局变量的第seq个元素，定时器超时后将从此数据结构中取出数据和参数。

在主线程中，程序会不断接收ACK recvseq，解析recvseq，如果recvseq==base则接收，设置PACK[recvseq+1].stoparr（关闭定时器），滑动窗口前进1；如果不等于base（base表示当前已发送未接收的最小序号，即第base个分组丢包），且!PACK[recvseq+1].cancelarr（定时器没有被取消，防止重复），则取消从base+1到seq的定时器（即除T_{base}的所有已发送包的定时器），设置T_{base}的targetseq=seq，即当T_{base}超时，重发从base到seq的所有包，并重新启动各自的定时器，此时注意要更新共享数据结构（主要是cancel和targetseq），防止产生无效的定时器。例如，当窗口大小为[9,14]时，收到了ACK 8（可能多个，使用!PACK[recvseq+1].cancelarr屏蔽后续，在重新启动前的超时时间内足以屏蔽），说明包9（到14）一定（至少有1个）没有传输成功，则取消10到14的所有定时器，设置定时器9的targetseq为14。等待9超时后，在定时器线程TimeThread重发9到14的所有包，并重启启动9到14的各自的定时器。

自此，直到收到所有ACK，关闭文件，退出发送模式。

七、修改（做了一半的）GBN为SR协议代码

实质上，由于SR对于定时器的控制比GBN简单（无需关闭后统一发送），因此无需统一管理包和定时器配置内容（无需手动关闭定时器），只需要对每个定时器传入包的信息作为参数即可。我在实现GBN时，在统一管理前考虑到SR可能会用到，使用Git备份了源代码，然后恰好SR用上了。如果要是直接修改GBN成品的话，可能还需要多费一些事情，以及运行时占用更多的内存。总体上来说，实现难度：发送端：GBN>SR，接收端：GBN<SR。

6.1 接收端

新建全局变量，用来存储乱序到达的数据，使用指针数组，接收到数据的时候再执行new字符数组操作，节省内存的同时，能通过扫描cachedBuffer判断是否为NULL判断有无缓存。

```
char *cachedBuffer[MAX_PACKAGE_NUM] = { 0 };
```

```
int cachedBufferSize[MAX_PACKAGE_NUM] = { 0 };
```

在初始化和连接建立后，用户在交互窗口使用“R”命令后，程序进入一个函数srrecv，同理先发LEN、等ACK0，然后循环等待DAT seq，遇到乱序的DAT保存到cachedBuffer里，注意保存前判断是否为NULL再new，防止保存重复；对于正确的seq则写到文件，然后base++，如果cachedBuffer[base]!=NULL，说明存在乱序缓存的包，继续写出、滑动，直到cachedBuffer[base]==NULL，发送ACK seq.....直到写到完整文件长度。

6.2 发送端

同理，发送端较为复杂，但没有统一的定时器管理，省了不少事。

进入发送模式后，仍然采用两个线程，一个线程一直发数据，主线程一直收ACK。定时器超时需要做修改，超时只发自己包就行了，不需要发别人的包。

八、引入随机丢包和打印优化

引入随机丢包的方式是采用随机数rand()，首先定义以下两个宏，分别表示开启随机丢包和丢包概率，丢包的概率的数值就是随机数小于这个值，则丢包。

```
#define ENABLE_MISSING true
```

```
#define MISSING_PR 1000 //max=32767
```

每次发送文件时，执行`srand((unsigned)time(NULL))`，重置随机数种子。

在各协议的接收端`recv`后，添加丢包代码：

```
if (ENABLE_MISSING) {  
    if (rand() < MISSING_PR) {  
        printf("模拟丢包! [%d]\n", recvseq);  
        continue;  
    }  
}
```

打印优化是指打印数据的格式进行优化。

1、添加数据报的输出。在数据发送和接收的统一出入口——`packData`和`unpackData`中添加`printf`，采用“C++”式的流式传输格式“<<<<”表示输出，“>>>>”表示输入，只打印控制字段和`seq`，Data部分使用长度代替。

例如：“<<<< SYN 10240 [17]”、“>>>> DATA 12 [502]”

2、添加进度条。添加函数`printStatus()`，在`recv`后，添加`printStatus()`调用。

每一个字符表示一个包，“#”表示接收成功，空格表示未接收的包，“-”表示已缓存（仅SR接收端）。后面再跟上已接收/总数字节和百分比。

```
void printStatus(int total, int curr, int totalb, bool currb[]) {  
    printf("[");  
    for (int i = 1; i <= totalb; i++) {  
        if (currb == NULL) {  
            if (PACK[i].stoparr)printf("#");  
            else printf(" ");  
        }  
        else {  
            if (currb[i] == true)printf("#");  
            else if (currb[i] == 2)printf("-");  
            else printf(" ");  
        }  
    }  
    printf("] %d/%d | %.2f%%\n", curr, total, curr / (double)total * 100.0);  
}
```

实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

下图就是一个模拟引入数据包的丢失、支持双向数据传输、一个 C/S 结构的文件传输应用。首先是 GBN 协议客户端：

```

10241 已连接至127.0.0.1:10239
初始化传输服务中...
命令行[0]: E:\OneDrive - stu.hit.edu.cn\学习\计算机网络\实验\HIT-CN-LAB3 (GBN&SR)\LAB3\Debug\GBNServer.exe
套接字将运行在端口: 10241
默认文件保存路径: ../DATA_10241
CreateProcess succeeded
初始化完成
WBXFileTransfer> S
正在启动传输, 文件将保存到文件夹: ../DATA_10241
正后台等待主机连接, 您也可以主动连接
WBXFileTransfer> WBXFileTransfer> >>>> SYN 10239 [31]
收到主机连接请求 127.0.0.1:10010239
正在连接主机 127.0.0.1:10239
<<<< SYN 0010241 [17]
连接成功: 127.0.0.1:10239

10239 已连接至127.0.0.1:10241
初始化传输服务中...
命令行[0]: PORT=10239
套接字将运行在端口: 10239
默认文件保存路径: ../DATA_10239
初始化完成
WBXFileTransfer> S
正在启动传输, 文件将保存到文件夹: ../DATA_10239
正后台等待主机连接, 您也可以主动连接
WBXFileTransfer> WBXFileTransfer> C
正在连接主机 127.0.0.1:10241
<<<< SYN 0010239 [17]
WBXFileTransfer> >>>> SYN 10241 [31]
收到主机连接请求 127.0.0.1:10010241
连接成功: 127.0.0.1:10010241
WBXFileTransfer>
    
```

用户打开两个程序, 输入S初始化保存位置, 输入C连接目标 (因为没有指定目标, 则使用默认目标: 本机测试)。经过两个SYN后, 两端显示连接成功。

然后左端用户输入R进行接收文件, 右端用户输入F发送文件 (没有指定发送的文件, 默认发送测试文件)。

首先发送了文件信息, 然后是数据。

```

10241 已连接至127.0.0.1:10239
连接成功: 127.0.0.1:10010239
WBXFileTransfer> R
正在等待接收, 文件将保存到文件夹: ../DATA_10241
>>>> LEN 20802 [32]
准备接收文件: transfer.10239.txt [大小: 0020802]
打开文件成功: ../DATA_10241/transfer.10239.txt
<<<< ACK 0000000 [0]
快速ACK成功
>>>> DAT 1 [514]
#####
<<<< ACK 0000001 [0]
>>>> DAT 2 [514]
#####
<<<< ACK 0000002 [0]
>>>> DAT 3 [514]
#####
<<<< ACK 0000003 [0]
>>>> DAT 4 [514]
#####
<<<< ACK 0000004 [0]
>>>> DAT 5 [514]
#####
<<<< ACK 0000005 [0]
>>>> DAT 6 [514]
#####
<<<< ACK 0000006 [0]
>>>> DAT 7 [514]
#####
<<<< ACK 0000007 [0]
>>>> DAT 8 [514]
#####
<<<< ACK 0000008 [0]
>>>> DAT 9 [514]
#####
<<<< ACK 0000009 [0]
>>>> DAT 10 [514]
#####
<<<< ACK 0000010 [0]
>>>> DAT 11 [514]
#####
<<<< ACK 0000011 [0]
>>>> DAT 12 [514]
#####
<<<< ACK 0000012 [0]
>>>> DAT 13 [514]
#####
<<<< ACK 0000013 [0]
>>>> DAT 14 [514]
#####
<<<< ACK 0000014 [0]
>>>> DAT 15 [514]
#####
#####
] 500/20802 | 2.40%
] 1000/20802 | 4.81%
] 1500/20802 | 7.21%
] 2000/20802 | 9.61%
] 2500/20802 | 12.02%
] 3000/20802 | 14.42%
] 3500/20802 | 16.83%
] 4000/20802 | 19.23%
] 4500/20802 | 21.63%
] 5000/20802 | 24.04%
] 5500/20802 | 26.44%
] 6000/20802 | 28.84%
] 6500/20802 | 31.25%
] 7000/20802 | 33.65%
] 7500/20802 | 36.05%

10239 已连接至127.0.0.1:10241
WBXFileTransfer> F
准备发送文件: ../DATA_10239/transfer.10239.txt
准备接收文件: transfer.10239.txt [大小: 20802]
<<<< LEN 0020802 [18]
>>>> ACK 0 [14]
窗口为: [1,6]
文件成功读入数据: 500
<<<< DAT 0000001 [500]
窗口为: [1,6]
文件成功读入数据: 500
启动定时器: DAT 1
<<<< DAT 0000002 [500]
窗口为: [1,6]
文件成功读入数据: 500
启动定时器: DAT 2
<<<< DAT 0000003 [500]
窗口为: [1,6]
文件成功读入数据: 500
启动定时器: DAT 3
<<<< DAT 0000004 [500]
窗口为: [1,6]
文件成功读入数据: 500
启动定时器: DAT 4
<<<< DAT 0000005 [500]
启动定时器: DAT 5
>>>> ACK 1 [14]
#####
] 500/20802 | 2.40%
>>>> ACK 2 [14]
#####
] 1000/20802 | 4.81%
>>>> ACK 3 [14]
#####
] 1500/20802 | 7.21%
>>>> ACK 4 [14]
#####
] 2000/20802 | 9.61%
>>>> ACK 5 [14]
#####
] 2500/20802 | 12.02%
窗口为: [6,11]
文件成功读入数据: 500
<<<< DAT 0000006 [500]
窗口为: [6,11]
文件成功读入数据: 500
启动定时器: DAT 6
<<<< DAT 0000007 [500]
>>>> ACK 6 [14]
#####
] 3000/20802 | 14.42%
启动定时器: DAT 7
窗口为: [6,11]
文件成功读入数据: 500
>>>> ACK 7 [14]
#####
] 3500/20802 | 16.83%
<<<< DAT 0000008 [500]
窗口为: [8,13]
文件成功读入数据: 500
    
```

下图中发生了模拟丢包, 左图显示GBN在窗口内的后续包都不被接受了, 并返回了最小的已接收的序号。右侧显示取消了所有定时器, 仅保留一个, 超时后将重发目标终点设置为已发送的最大值。

The screenshot displays a network simulation window with two panes. The left pane shows a log of packet transmissions between two hosts (10241 and 10239). The right pane shows the status of various timers and acknowledgments. Red arrows indicate the flow of control: when a packet is received or an acknowledgment is sent, a timer is started or reset. For example, receiving packet 16 starts a timer, and receiving acknowledgment 15 resets it. The log shows packets 10 through 24 being transmitted successfully, with corresponding acknowledgments received.

在第一个定时器超时后，重发了所有的包，并分别启动了新的定时器。

This screenshot shows the simulation after a timeout event. The left pane shows that packets 17 through 24 were retransmitted. The right pane shows the status of the timers, indicating that they have expired and new timers have been started for the retransmitted packets. Red arrows highlight the retransmission of packets 17, 18, 19, and 20, and the corresponding timer actions. The simulation continues to show successful transmission of the remaining packets and acknowledgments.

文件传输完成！双方用户还可以再输入控制命令来选择发送或接收。

下面启动SR协议客户端。

可以看到，当发生了丢包，后续的包都传了过来，进行了缓存。
收到了丢失的包后，缓存写出，窗口大步滑动。

很不幸，用户在发最后一个包的时候，又发生了丢包！不过没有关系我们可以重传。

问题讨论：

1、在实现的过程中，发现关于Winsock返回的错误信息的问题：

由于一开始我在packData中的疏忽，将多余的“\0”进入到winsock中发送，在unpackData中导致了后续中发生“Message too long”的问题。因为“\0”也算作一个字符，而Buffer不足，则recv就会返回这个错误，这是我经过1小时的单步调试发现的问题，当时发现所有的字符已经正常recv，后面也存在“\0”，后来才发现，发送的数据中后面还存在“\0”。经过数值调整，已修复。

2、正如正文所说，我在实现GBN时，在统一管理前考虑到SR可能会用到，使用Git备份了源代码，然后恰好SR用上了，说明常常commit还是十分有用的。

3、总体上来说，实现难度：发送端：GBN>SR，接收端：GBN<SR。

心得体会：

结合实验过程和结果给出实验的体会和收获。

连肝3天，报告和代码纯手敲，绝无CV！

详细了解和学会了GBN、SR的可靠数据传输的思想。