计算机网络概论_第一次实验——滑动窗口

实验流程

1、 配置实验环境
2、 阅读 netRiver 实验指导书中关于滑动窗口部分的介绍。重点包括：frame 数据结构，frame_head 数据结构，frame_kind 枚举；三种滑动窗口测试函数在什么情况下被调用，以及传入的参数的意义，需要处理的情况（三种，包括 timeout，send，receive。其中 receive 还有 NAK 和 ACK 的区分），在各种情况下的处理要求。
3、 根据指导书中滑动窗口部分的介绍完成函数
4、 反复调试，直到成功通过测试

问题

1、 实验中遇到的第一个问题是实验系统要求的滑动窗口和老师上课讲的不一样。在该实验中，序号空间从 1 开始编号，一开始让 lower 和 upper 都是 0，超时帧无法重发并且在 ACK 时会多释放一个不该释放的 frame。并且不知道自己的程序哪里出问题，去网上搜索时有建议把调试关键信息（即由服务器给的帧信息）写入文件，方便调试。通过写入文件的方法终于发现序号空间从 1 开始编号。
2、 实验中遇到的第二个问题是回退 N 的滑动窗口，在 timeout 时，按照教学网实验指导所写，如果 1、2、3、4、5 帧中 3 超时，则 3、4、5 重发，实际上要求 1、2、3、4、5 均重发。这个问题看到群里同学的讨论才知道情况。

解决方法

如问题 1 中提到，通过把一些代码编写者不知道的信息写入文件可以方便整个调试流程。

代码

```
// 1500012945 Yuzhao Jiang
#include "sysinclude.h"
#include <cstring>
#include <iostream>
#include <fstream>
#include <queue>

extern void SendFRAMEPacket(unsigned char* pData, unsigned int len);

#define WINDOW_SIZE_STOP_WAIT 1
#define WINDOW_SIZE_BACK_N_FRAME 4

// native struct define
typedef enum {DATA,ACK,NAK} frame_kind;
typedef struct frame_head
{
    frame_kind kind; //帧类型
    unsigned int seq; //序列号
    unsigned int ack; //确认号
```

```
    unsigned char data[100]; //数据
};
typedef struct frame
{
    frame_head head; //帧头
    unsigned int size; //数据的大小
};

/*
* 停等
*/

int stud_slide_window_stop_and_wait(char *pBuffer, int bufferSize, UINT8 messageType)
{
    static frame* frameWindow[WINDOW_SIZE_STOP_WAIT];
    static int windowFrameLen[WINDOW_SIZE_STOP_WAIT];
    // when you see debug file, you will find the frame's seq is begin from 1.
    // so if you assign upper and lower 0, when you do accept,
    // you will free something shouldn't be free. You had better make upper and lower
    // correspond to the frame the experimental system sent to you.
    static int upper = 1, lower = 1;
    static queue<frame *> waitingFrame;
    static queue<int> waitingFrameLen;
    static ofstream out("stopWait.txt");

    // temp var table
    frame_head *header = (frame_head *)pBuffer;
    int seq = ntohl(header->seq);
    int ack = ntohl(header->ack);
    frame_kind kind = (frame_kind)(ntohl((u_long)header->kind));

    if (messageType == MSG_TYPE_TIMEOUT){
        int timeoutSeq = ntohl(*( (unsigned int *)pBuffer ));

        out << "超时/n" <<" timeoutseq = "<< timeoutSeq <<endl;
        out << "全部重发"<<endl;
        out.flush();

        for (int i = lower; i < upper; ++i){
            SendFRAMEPacket((unsigned char *)frameWindow[i %
WINDOW_SIZE_STOP_WAIT],
                    windowFrameLen[i % WINDOW_SIZE_STOP_WAIT]);
        }
    }
```

```
    else if(messageType == MSG_TYPE_SEND){
        //log package
        out << "发送/n" << "kind = " << kind <<" seq = "<<seq<<" ack = "<<ack<<" size =
"<<bufferSize<<endl;

        frame *frameToWait = new frame;
        *frameToWait = *(frame *)pBuffer;
        // window full
        if (upper - lower >= WINDOW_SIZE_STOP_WAIT){
            waitingFrame.push(frameToWait);
            waitingFrameLen.push(bufferSize);

            out<<"入队等待"<<endl;
            out.flush();
        }
        // window not full
        else{
            frameWindow[upper % WINDOW_SIZE_STOP_WAIT] = frameToWait;
            windowFrameLen[upper % WINDOW_SIZE_STOP_WAIT] = bufferSize;
            ++upper;
            SendFRAMEPacket((unsigned char *)pBuffer, bufferSize);

            out<<"直接发送"<<endl;
            out.flush();
        }
    }
    else if (messageType == MSG_TYPE_RECEIVE){
        //log package
        out << "接收/n" << "kind = " << kind <<" seq = "<<seq<<" ack = "<<ack<<endl;
        if (ack < lower || ack >= upper)
            return 1;
        if (kind == NAK){
            // deny the frame
            //log package
            out << "否定确认帧，重发"<<endl;
            out.flush();
            // maybe i can reach ack
            for (int i = lower; i < ack; ++i){
                SendFRAMEPacket((unsigned char *)frameWindow[i %
WINDOW_SIZE_STOP_WAIT], windowFrameLen[i % WINDOW_SIZE_STOP_WAIT]);
            }
            // while(ack<upper){
```

```
                //    SendFRAMEPacket((unsigned
char*)frameBuffer[ack%WINDOW_SIZE_STOP_WAIT],frameLenBuffer[ack%WINDOW_SIZE_STOP
_WAIT]);
                // }
                return 0;
            }
            // kind := ACK
            // lower may not reach ack
            for (; lower <= ack; ++lower){
                out << "accept " << lower << " package." << endl;
                delete frameWindow[lower % WINDOW_SIZE_STOP_WAIT];
            }

            while (!waitingFrame.empty() && upper - lower < WINDOW_SIZE_STOP_WAIT){
                frameWindow[upper % WINDOW_SIZE_STOP_WAIT] = waitingFrame.front();
                waitingFrame.pop();
                windowFrameLen[upper % WINDOW_SIZE_STOP_WAIT] =
waitingFrameLen.front();
                waitingFrameLen.pop();
                SendFRAMEPacket((unsigned char *)frameWindow[upper %
WINDOW_SIZE_STOP_WAIT],
                    windowFrameLen[upper % WINDOW_SIZE_STOP_WAIT]);
                ++upper;
            }
        }
        return 0;
}

/*
*  回退 N
*/

int stud_slide_window_back_n_frame(char *pBuffer, int bufferSize, UINT8 messageType)
{
        static frame* frameWindow[WINDOW_SIZE_BACK_N_FRAME];
        static int windowFrameLen[WINDOW_SIZE_BACK_N_FRAME];
        static int upper = 1, lower = 1;
        static queue<frame *> waitingFrame;
        static queue<int> waitingFrameLen;
        static ofstream out("backN.txt");

        // temp var table
        frame_head *header = (frame_head *)pBuffer;
        int seq = ntohl(header->seq);
```

```
    int ack = ntohl(header->ack);
    frame_kind kind = (frame_kind)(ntohl((u_long)header->kind));

    if (messageType == MSG_TYPE_TIMEOUT){
        int timeoutSeq = ntohl(*( (unsigned int *)pBuffer ));

        out << "超时/n" <<" timeoutseq = "<< timeoutSeq <<endl;
        out << "全部重发"<<endl;
        out.flush();

        for (int i = lower; i < upper; ++i){
            SendFRAMEPacket((unsigned char *)frameWindow[i %
WINDOW_SIZE_BACK_N_FRAME],
                    windowFrameLen[i % WINDOW_SIZE_BACK_N_FRAME]);
        }
    }
    else if(messageType == MSG_TYPE_SEND){
        //log package
        out << "发送/n" << "kind = " << kind <<" seq = "<<seq<<" ack = "<<ack<<" size =
"<<bufferSize<<endl;

        frame *frameToWait = new frame;
        *frameToWait = *(frame *)pBuffer;
        // window full
        if (upper - lower >= WINDOW_SIZE_BACK_N_FRAME){
            waitingFrame.push(frameToWait);
            waitingFrameLen.push(bufferSize);

            out<<"入队等待"<<endl;
            out.flush();
        }
        // window not full
        else{
            frameWindow[upper % WINDOW_SIZE_BACK_N_FRAME] = frameToWait;
            windowFrameLen[upper % WINDOW_SIZE_BACK_N_FRAME] = bufferSize;
            ++upper;
            SendFRAMEPacket((unsigned char *)pBuffer, bufferSize);

            out<<"直接发送"<<endl;
            out.flush();
        }
    }
    else if (messageType == MSG_TYPE_RECEIVE){
        //log package
```

```
            out << "接收/n" << "kind = " << kind <<" seq = "<<seq<<" ack = "<<ack<<endl;
            if (ack < lower || ack >= upper)
                return 1;
            if (kind == NAK){
                // deny the frame
                //log package
                out << "否定确认帧，重发"<<endl;
                out.flush();
                // maybe i can reach ack
                for (int i = lower; i < ack; ++i){
                    SendFRAMEPacket((unsigned char *)frameWindow[i %
WINDOW_SIZE_BACK_N_FRAME], windowFrameLen[i % WINDOW_SIZE_BACK_N_FRAME]);
                }
                // while(ack<upper){
                //     SendFRAMEPacket((unsigned
char*)frameBuffer[ack%WINDOW_SIZE_BACK_N_FRAME],frameLenBuffer[ack%WINDOW_SIZE_
BACK_N_FRAME]);
                // }
                return 0;
            }
            // kind := ACK
            // lower may not reach ack
            for (; lower <= ack; ++lower){
                out << "accept " << lower << " package." << endl;
                delete frameWindow[lower % WINDOW_SIZE_BACK_N_FRAME];
            }

            while (!waitingFrame.empty() && upper - lower < WINDOW_SIZE_BACK_N_FRAME){
                frameWindow[upper % WINDOW_SIZE_BACK_N_FRAME] = waitingFrame.front();
                waitingFrame.pop();
                windowFrameLen[upper % WINDOW_SIZE_BACK_N_FRAME] =
waitingFrameLen.front();
                waitingFrameLen.pop();
                SendFRAMEPacket((unsigned char *)frameWindow[upper %
WINDOW_SIZE_BACK_N_FRAME],
                    windowFrameLen[upper % WINDOW_SIZE_BACK_N_FRAME]);
                ++upper;
            }
        }
        return 0;
}

/*
* 选择重传
```

```
*/
int stud_slide_window_choice_frame_resend(char *pBuffer, int bufferSize, UINT8 messageType)
{
	static frame* frameWindow[WINDOW_SIZE_STOP_WAIT];
	static int windowFrameLen[WINDOW_SIZE_STOP_WAIT];
	static int upper = 1, lower = 1;
	static queue<frame *> waitingFrame;
	static queue<int> waitingFrameLen;
	static ofstream out("select.txt");

	// temp var table
	frame_head *header = (frame_head *)pBuffer;
	int seq = ntohl(header->seq);
	int ack = ntohl(header->ack);
	frame_kind kind = (frame_kind)(ntohl((u_long)header->kind));

	if (messageType == MSG_TYPE_TIMEOUT){
		int timeoutSeq = ntohl(*( (unsigned int *)pBuffer ));

		out << "超时/n" <<" timeoutseq = "<< timeoutSeq <<endl;
		out << "重发超时帧"<<endl;
		out.flush();
		// resend directly
		SendFRAMEPacket((unsigned char *)frameWindow[timeoutSeq %
WINDOW_SIZE_BACK_N_FRAME],
			windowFrameLen[timeoutSeq % WINDOW_SIZE_BACK_N_FRAME]);
		return 0;
	}
	else if(messageType == MSG_TYPE_SEND){
		//log package
		out << "发送/n" << "kind = " << kind <<" seq = "<<seq<<" ack = "<<ack<<" size =
"<<bufferSize<<endl;

		frame *frameToWait = new frame;
		*frameToWait = *(frame *)pBuffer;
		// window full
		if (upper - lower >= WINDOW_SIZE_BACK_N_FRAME){
			waitingFrame.push(frameToWait);
			waitingFrameLen.push(bufferSize);

			out<<"入队等待"<<endl;
			out.flush();
		}
		// window not full
```

```
            else{
                frameWindow[upper % WINDOW_SIZE_BACK_N_FRAME] = frameToWait;
                windowFrameLen[upper % WINDOW_SIZE_BACK_N_FRAME] = bufferSize;
                ++upper;
                SendFRAMEPacket((unsigned char *)pBuffer, bufferSize);

                out<<"直接发送"<<endl;
                out.flush();
            }
        }
        else if (messageType == MSG_TYPE_RECEIVE){

            if( ack<lower || ack >=upper )
                return 1;

            if (kind == NAK){
                // resend a frame
                out << "重发一帧"<<endl;
                out.flush();

                SendFRAMEPacket((unsigned char *)frameWindow[ack %
WINDOW_SIZE_BACK_N_FRAME],
                        windowFrameLen[ack % WINDOW_SIZE_BACK_N_FRAME]);
                return 0;
            }
            // kind := ACK
            // lower may not reach ack(ack may not need to free)
            for (; lower <= ack; ++lower){
                out << "accept " << lower << " package." << endl;
                delete frameWindow[lower % WINDOW_SIZE_BACK_N_FRAME];
            }

            while (!waitingFrame.empty() && upper - lower < WINDOW_SIZE_BACK_N_FRAME){
                frameWindow[upper % WINDOW_SIZE_BACK_N_FRAME] = waitingFrame.front();
                waitingFrame.pop();
                windowFrameLen[upper % WINDOW_SIZE_BACK_N_FRAME] =
waitingFrameLen.front();
                waitingFrameLen.pop();
                SendFRAMEPacket((unsigned char *)frameWindow[upper %
WINDOW_SIZE_BACK_N_FRAME],
                        windowFrameLen[upper % WINDOW_SIZE_BACK_N_FRAME]);
                ++upper;
            }
```

```
        }
        return 0;
}
```