滑动窗口协议实验

1600013019 张智涵

一、实验内容

本实验要求实现一个数据链路层协议的网络传输部分，通过C语言编程模拟滑动窗口协议的用户端，和服务器进行数据传输和确认。实验共分为三个子实验，分别实现停等协议、回退N滑动窗口协议和选择重传协议。

滑动窗口协议的基本原理是收发双方分别维护一个长度固定的接收/发送窗口，只能接收/发送在窗口范围内的帧。用户发送帧之后启动计时器，服务器接收帧之后发送ACK确认帧给用户端；如果过长时间内没有收到ACK，发送端重新传送窗口内的帧。在选择重传协议中，发送错误的帧序号会通过NAK帧传回，发送端需要重新发送错误的那一帧。

本实验中要求实现滑动窗口协议中的用户端部分，即发送提供的帧到服务器、接收服务器的ACK帧并改变滑动窗口、接收服务器的NAK帧并重传错误帧（选择重传协议）、接收超时信息并重传发送窗口内的帧。

二、编程实现

由于子实验1：停等协议实验和子实验2：回退N滑动窗口协议实验在发送端的唯一区别是发送窗口长度不同（停等协议：窗口长度为1），因此编程时直接实现了回退N滑动窗口协议，在做停等协议实验时直接复制了回退N协议的代码并且更改了窗口长度。

1.回退N滑动窗口协议实验

首先根据messageType参数将函数分为三个分支：发送帧、接收ACK、处理超时。同时维护全局数组buffer作为发送队列、两个全局变量top和bottom作为滑动窗口的上下界。

在发送分支中，传入函数的参数pBuffer应当是需要发送的帧。先将pBuffer存入buffer数组中，再判断发送窗口有没有打开到上限（即top-bottom和WINDOW_SIZE的大小关系）。如果打开到上限，则直接返回；否则，调用SendFRAMEPacket函数发送这一帧，并将top加1，表明新打开了一个窗口。

在接收分支中，传入函数的参数pBuffer应当是ACK确认帧。根据pBuffer->ack和bottom的差值ack_cnt，可以算出这次ACK帧确认了多少个帧的正确传输，将top和bottom都加上ack_cnt，相当于把发送窗口前移ack_cnt个单位，并且调用SendFRAMEPacket函数将新进入窗口的ack_cnt个帧发送给服务器。

在超时分支中，直接调用SendFRAMEPacket函数WINDOW_SIZE次，将发送窗口内的帧重传给服务器。

2.停等协议实验

同回退N协议实验，只是将WINDOW_SIZE设为1。

3.选择重传协议实验

选择重传协议的实现函数分为三个分支：发送分支、接收ACK分支和接收NAK分支。其中发送分支和接收ACK分支分别与回退N协议实验中发送分支和接收分支的实现相同。

在接收NAK分支中，读出NAK帧中的ack序号，从buffer数组中找到这一帧并且调用SendFRAMEPacket函数发送给服务器即可。

实验代码：

```
#include "sysinclude.h"
#include <iostream>
#include <vector>
#include <stdlib.h>
#include <cstring>
#include <winsock.h>
using namespace std;

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

#define WINDOW_SIZE_STOP_WAIT 1
#define WINDOW_SIZE_BACK_N_FRAME 4

typedef enum {DATA, ACK, NAK} frame_kind;
typedef struct frame_head
{
    unsigned int kind;
    unsigned int seq;
    unsigned int ack;
    char data[100];
};
typedef struct frame
{
    frame_head head;
    unsigned int size;
};

vector <char*> buffer; // sending queue buffer
int top = 0, bottom = 0; // window boundaries

/*
* 停等协议测试函数
*/
int stud_slide_window_stop_and_wait(char *pBuffer, int bufferSize, UINT8 messageType)
{
    frame_head *fr = new(frame_head);
    memcpy(fr, pBuffer, bufferSize);

    if (messageType == MSG_TYPE_SEND) {
        if (ntohl(fr -> seq) == 1) {
            buffer.clear();
            top = 0; bottom = 0;
        }
        int width = top - bottom;
        buffer.push_back((char*)fr);

        if (width >= WINDOW_SIZE_STOP_WAIT)
            return 0;
        top++; //open a window
        SendFRAMEPacket((unsigned char *)pBuffer, bufferSize);
    }

    else if (messageType == MSG_TYPE_RECEIVE) {
        int ack_cnt = ntohl(fr -> ack) - bottom; // how many frames have been received
        bottom += ack_cnt;
```

```
        if (buffer.size() >= top + 1) { // there are waiting frames to send
            for (int j = top; j < top + ack_cnt; j++) {
                frame_head *wait_fr = (frame_head *) buffer[j];
                SendFRAMEPacket((unsigned char *)buffer[j], strlen((wait_fr -> data)) + 13);
            }
            top += ack_cnt;
        }
    }

    else if (messageType == MSG_TYPE_TIMEOUT) {
        for (int j = bottom; j < top; j++) {
            frame_head *wait_fr = (frame_head *) buffer[j];
            SendFRAMEPacket((unsigned char *)buffer[j], strlen((wait_fr -> data)) + 13);
        }

    }

    return 0;


}

/*
* 回退n帧测试函数
*/
int stud_slide_window_back_n_frame(char *pBuffer, int bufferSize, UINT8 messageType)
{
    frame_head *fr = new(frame_head);
    memcpy(fr, pBuffer, bufferSize);

    if (messageType == MSG_TYPE_SEND) {
        if (ntohl(fr -> seq) == 1) {
            buffer.clear();
            top = 0; bottom = 0;
        }
        int width = top - bottom;
        buffer.push_back((char*)fr);

        if (width >= WINDOW_SIZE_BACK_N_FRAME)
            return 0;
        top++; //open a window
        SendFRAMEPacket((unsigned char *)pBuffer, bufferSize);
    }

    else if (messageType == MSG_TYPE_RECEIVE) {
        int ack_cnt = ntohl(fr -> ack) - bottom; // how many frames have been received
        bottom += ack_cnt;

        if (buffer.size() >= top + 1) { // there are waiting frames to send
            for (int j = top; j < top + ack_cnt; j++) {
                frame_head *wait_fr = (frame_head *) buffer[j];
                SendFRAMEPacket((unsigned char *)buffer[j], strlen((wait_fr -> data)) + 13);
            }
            top += ack_cnt;
        }
    }

    else if (messageType == MSG_TYPE_TIMEOUT) {
        for (int j = bottom; j < top; j++) {
            frame_head *wait_fr = (frame_head *) buffer[j];
```

```
            SendFRAMEPacket((unsigned char *)buffer[j], strlen((wait_fr -> data)) + 13);
        }

    }

    return 0;

}

/*
* 选择性重传测试函数
*/
int stud_slide_window_choice_frame_resend(char *pBuffer, int bufferSize, UINT8 messageType)
{
    frame_head *fr = new(frame_head);
    memcpy(fr, pBuffer, bufferSize);

    if (messageType == MSG_TYPE_SEND) {
        if (ntohl(fr -> seq) == 1) {
            buffer.clear();
            top = 0; bottom = 0;
        }
        int width = top - bottom;
        buffer.push_back((char*)fr);

        if (width >= WINDOW_SIZE_BACK_N_FRAME)
            return 0;
        top++; //open a window
        SendFRAMEPacket((unsigned char *)pBuffer, bufferSize);
    }

    else if (messageType == MSG_TYPE_RECEIVE && ntohl(fr -> kind) == ACK) {
        int ack_cnt = ntohl(fr -> ack) - bottom; // how many frames have been received
        bottom += ack_cnt;

        if (buffer.size() >= top + 1) { // there are waiting frames to send
            for (int j = top; j < top + ack_cnt; j++) {
                frame_head *wait_fr = (frame_head *) buffer[j];
                SendFRAMEPacket((unsigned char *)buffer[j], strlen((wait_fr -> data)) + 13);
            }
            top += ack_cnt;
        }
    }

    else if (messageType == MSG_TYPE_RECEIVE && ntohl(fr -> kind) == NAK) {
        int nak_num = ntohl(fr -> ack) - 1;
        frame_head *wait_fr = (frame_head *) buffer[nak_num];
        SendFRAMEPacket((unsigned char *)buffer[nak_num], strlen((wait_fr -> data)) + 13);

    }

    return 0;
}
```