**滑动窗口协议实验上机报告**

1. **实验目的**

实现一个数据链路层协议的数据传送部分，理解数据链路层协议中的“滑动窗口”技术的基本工作原理，掌握计算机网络协议的基本实现技术。

1. **实验要求**

在一个数据链路层的模拟实现环境中，用 C 语言实现下面三个数据链路层协议。

1）1 比特滑动窗口协议

2）回退 N 帧滑动窗口协议

3）选择性重传协议

1. **实验内容**

根据滑动窗口协议，模拟滑动窗口协议中发送端的功能，对系统发送的帧缓存并加入窗口等待确认，并在超时或者错误时对部分帧进行重传。

编写停等及退回 N 滑动窗口协议函数，响应系统的发送请求、接收帧消息以及超时消息，并根据滑动窗口协议进行相应处理。

编写选择性重传协议函数，响应系统的发送请求、接受帧消息以及错误消息，并根据滑动窗口协议进行相应处理。

1. **实验思路**

缓存和窗口都是先进先出的结构，使用队列的数据结构来处理缓存及滑动窗口，queue<struct frame> sendList 来缓存发送信息，用deque<struct frame> sendWindow 来表示滑动窗口。

#include "sysinclude.h"

#include<queue>

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

{

frame\_kind kind;

unsigned int seq;

unsigned int ack;

unsigned char data[100];

};

typedef struct frame

{

frame\_head head;

unsigned int size;

};

queue<struct frame> sendList; //发送列表

deque<struct frame> sendWindow; //滑动窗口

/\*

\* 停等协议测试函数

\*/

int stud\_slide\_window\_stop\_and\_wait(char \*pBuffer, int bufferSize, UINT8 messageType)

{

static bool send = true; // 记录当前状态是否可以发送

struct frame f;

switch(messageType)

{

case MSG\_TYPE\_SEND://上层下达发送指令

memcpy(&f, pBuffer, sizeof(f)); //待发送帧

f.size = bufferSize;

sendList.push(f); //将帧加入发送队列

if(send) //如果可以发送，从发送队列中取出一帧，直接发送，然后将发送状态置为否，等待接收方确认信息

{

f = sendList.front();

SendFRAMEPacket((unsigned char\*)(&f), f.size);

send = false;

}

break; //如果还未受到接收方的确认，则缓存后直接跳出

case MSG\_TYPE\_RECEIVE: //接收到确认帧

sendList.pop(); //将发送队列中已发送的内容删去

send = true; //可以发送下一帧

if(!sendList.empty()) //如果还有内容要发送

{

f=sendList.front(); //从发送列表中取出一帧

SendFRAMEPacket((unsigned char\*)(&f), f.size);

send = false; //发送状态置为否，等待下一个确认

}

break;

case MSG\_TYPE\_TIMEOUT:

f = sendList.front(); //重新发送上一帧

SendFRAMEPacket((unsigned char\*)(&f), f.size);

send = false; //等待确认

break;

default:

break;

}

return 0;

}

/\*

\* 回退n帧测试函数

\*/

int stud\_slide\_window\_back\_n\_frame(char \*pBuffer, int bufferSize, UINT8 messageType)

{

struct frame f;

switch(messageType)

{

case MSG\_TYPE\_SEND:

memcpy(&f,pBuffer,sizeof(f));

f.size = bufferSize;

sendList.push(f); //将待发送帧加入发送队列

if(sendWindow.size() < WINDOW\_SIZE\_BACK\_N\_FRAME) // 如果发送窗口未满，将该帧加入发送队列，并发送该帧

{

f = sendList.front();

sendWindow.push\_back(f);

SendFRAMEPacket((unsigned char\*)(&f),f.size);

sendList.pop();

}

break;

case MSG\_TYPE\_RECEIVE: //接收到确认帧

memcpy(&f,pBuffer,sizeof(f));

while(ntohl(sendWindow.begin()->head.seq) != ntohl(f.head.ack) && !sendWindow.empty()) //将确认帧的seq之前的帧全部从发送列表删除，因为已经到达

{

sendWindow.pop\_front();

}

sendWindow.pop\_front();

//如果有可发送的缓存，且滑动窗口未满，则发送

while(sendWindow.size()<WINDOW\_SIZE\_BACK\_N\_FRAME && !sendList.empty())

{

f=sendList.front();

sendWindow.push\_back(f);

sendList.pop();

SendFRAMEPacket((unsigned char\*)(&f),f.size);

}

break;

case MSG\_TYPE\_TIMEOUT: //超时，则将发送窗口中的全部内容发送

for(deque<struct frame>::iterator iter = sendWindow.begin(); iter != sendWindow.end(); ++ iter)

{

SendFRAMEPacket((unsigned char\*)&(\*iter),iter->size);

}

break;

default:

break;

}

}

/\*

\* 选择性重传测试函数

\*/

int stud\_slide\_window\_choice\_frame\_resend(char \*pBuffer, int bufferSize, UINT8 messageType)

{

struct frame f;

switch(messageType)

{

case MSG\_TYPE\_SEND:

memcpy(&f,pBuffer,sizeof(f));

f.size = bufferSize;

sendList.push(f); //加入待发送列表

if(sendWindow.size() < WINDOW\_SIZE\_BACK\_N\_FRAME) //如果发送窗口未满则发送

{

f=sendList.front();

sendWindow.push\_back(f);

sendList.pop();

SendFRAMEPacket((unsigned char\*)(&f),f.size);

}

break;

case MSG\_TYPE\_RECEIVE:

memcpy(&f,pBuffer,sizeof(f));

if(ntohl(f.head.kind) == ack) //若接收到的是确认信息

{

while(ntohl(sendWindow.begin()->head.seq) != ntohl(f.head.ack) && !sendWindow.empty()) //则将发送窗口的末端移到确认帧之后

{

sendWindow.pop\_front();

}

sendWindow.pop\_front();

}

else if(ntohl(f.head.kind) == nak) //如果收到的是nak，则需要选择重传

{

for(deque<struct frame>::iterator iter = sendWindow.begin(); iter != sendWindow.end(); ++ iter)

{

if(ntohl(f.head.ack)==ntohl(iter->head.seq)) //从发送窗口中找到发送失败的帧，重新发送

{

SendFRAMEPacket((unsigned char\*)&(\*iter),iter->size);

break;

}

}

}

while(sendWindow.size()<WINDOW\_SIZE\_BACK\_N\_FRAME && !sendList.empty()) //如果发送窗口未满，则发送新的内容

{

f=sendList.front();

sendWindow.push\_back(f);

sendList.pop();

SendFRAMEPacket((unsigned char\*)(&f),f.size);

}

break;

case MSG\_TYPE\_TIMEOUT: //超时

unsigned int seq;

memcpy(&seq,pBuffer,sizeof(seq));

//将超时的帧重新发送

for(deque<struct frame>::iterator iter = sendWindow.begin(); iter != sendWindow.end(); ++ iter)

{

if(ntohl(seq) == ntohl(iter->head.seq))

{

SendFRAMEPacket((unsigned char\*)(&(\*iter)),iter->size);

break;

}

}

break;

default:

break;

}

return 0;

}