**滑动窗口协议实验上机报告**

姓 名： 肖 威

学 号： 1400011059

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**

#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;

deque<struct frame> sendWindow;

/\*

\* 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;

struct frame f;

switch(messageType)

{

case MSG\_TYPE\_SEND:

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

f.size = bufferSize;

sendList.push(f);

if(send)

{

f = sendList.front();

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

send = false;

}

break;

case MSG\_TYPE\_RECEIVE:

sendList.pop();

send = true;

if(!sendList.empty())

{

f=sendList.front();

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

send = false;

}

break;

case MSG\_TYPE\_TIMEOUT:

f = sendList.front();

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

send = false;

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);

if(sendWindow.size() < WINDOW\_SIZE\_BACK\_N\_FRAME)

{

f = sendList.front();

sendWindow.push\_back(f);

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

sendList.pop();

}

break;

case MSG\_TYPE\_RECEIVE:

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

while(ntohl(sendWindow.begin()->head.seq) != ntohl(f.head.ack) && !sendWindow.empty())

{

sendWindow.pop\_front();

}

sendWindow.pop\_front();

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:

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);

if(sendWindow.size() < WINDOW\_SIZE\_BACK\_N\_FRAME)

{

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)

{

while(ntohl(sendWindow.begin()->head.seq) != ntohl(f.head.ack) && !sendWindow.empty())

{

sendWindow.pop\_front();

}

sendWindow.pop\_front();

}

else if(ntohl(f.head.kind) == nak)

{

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

{

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

{

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

break;

}

}

}

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:

unsigned int seq;

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

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;

}