2020/08/12_网络编程_第6课_RAW套接字抓包、WireShark的使用、异步套接字服务器

笔记本:	网络编程
创建时间:	2020/8/12 星期三 10:30
作者:	ileemi

原始套接字 -- RAW

RAW套接字抓包

研究他人协议:

1. 抓包 -- 可以监听，不可以改包，不能判断是哪个程序发送的包

将网卡设置为混杂模式，不区分IP地址和端口 API -- **WSAIoctl**

判断相关协议:
显示相应的信息 -- 源端口，目标端口，校验和，数据长度，数据等（协议中数据输出的时候需要记得转换大小尾）

使用原始套接字进行抓包（网络层进行拦截），代码示例:

```
// ICMP.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//POP3 STMP 邮箱协议

#include <stdio.h>
#include <Winsock2.h>
#include <ws2tcpip.h>
#include <Mstcpip.h>
#pragma comment(lib, "Ws2_32.lib")
SOCKET g_Socket;
class CSocket {
public:
    CSocket() {
        //初始化套接字库
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            printf("
[server] WSAStartup error:%08X\n", WSAGetLastError());
        }
```

```
    }
    ~CSocket()  {
        //反初始化库
        WSACleanup();
    }
}g_init;
void  show_error_msg(const  char*  pre)  {
    LPVOID  lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER  |
        FORMAT_MESSAGE_FROM_SYSTEM  |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL,  SUBLANG_DEFAULT),  // Default  language

        (LPTSTR)&  lpMsgBuf,
        0,
        NULL
    );
    printf("%s:%s",  pre,  (LPCTSTR)lpMsgBuf);
    // Free  the  buffer.
    LocalFree(lpMsgBuf);
}
struct  ip_hdr
{
    unsigned  char  h_len  :  4;              //length  of  header
    unsigned  char  version  :  4;          //Version  of  IP
    unsigned  char  tos;                          //Type  of  service
    unsigned  short  total_len;           //total  length  of  the  packet

    unsigned  short  ident;                    //unique  identifier
    unsigned  short  frag_and_flags;  //flags
    unsigned  char  ttl;                          //ttl
    unsigned  char  proto;                     //protocol(TCP  ,UDP  etc)

    unsigned  short  checksum;              //IP  checksum
    unsigned  int  sourceIP;
    unsigned  int  destIP;
};
#define  ECHO_REQUEST    8
#define  ECHO_REPLY        0
struct  icmp_hdr
{
    unsigned  char    icmp_type;      //类型
    unsigned  char    icmp_code;      //代码
    unsigned  short  icmp_cksum;    //效验和
```

```
    unsigned  short  icmp_id;          //n
    unsigned  short  icmp_seq;        //n
    unsigned  int      icmp_data;      //GetTickout()
};
struct  udp_header  {
    unsigned  short  src_port;  //  源端口号16bit
    unsigned  short  dst_prot;  //  目的端口号16bit
    unsigned  short  len;            //  数据包长度16bit
    unsigned  short  check_sum;//  校验和16bit
};
typedef  struct  tcp_header
{
    unsigned  short  nSourPort;      //  源端口号16bit
    unsigned  short  nDestPort;      //  目的端口号16bit
    unsigned  int      nSequNum;        //  序列号32bit
    unsigned  int      nAcknowledgeNum;//  确认号32bit
    unsigned  short  nHLenAndFlag;      //  前4位：TCP头长度；中6位：保
留；后6位：标志位16bit
    unsigned  short  nWindowSize;        //  窗口大小16bit
    unsigned  short  nCheckSum;            //  检验和16bit
    unsigned  short  nrgentPointer;    //  紧急数据偏移量16bit
};
//网际校验和是被校验数据16位值的反码和(ones-complement  sum)
WORD  CalcCheckSum(IN  unsigned  short*  addr,  IN  int  len)
{
    int      nleft  =  len;
    int      sum  =  0;
    unsigned  short*  w  =  addr;
    unsigned  short  answer  =  0;
    while  (nleft  >  1)  {
        sum  +=  *w++;
        nleft  -=  2;
    }
    if  (nleft  ==  1)  {
        *(unsigned  char*)(&answer)  =  *(unsigned  char*)w;
        sum  +=  answer;
    }
    sum  =  (sum  >>  16)  +  (sum  &  0xffff);
    sum  +=  (sum  >>  16);
    answer  =  ~sum;
    return  (answer);
}
//抓包
int  main()
{
    //  1.初始化套接字(说明使用的协议)
    g_Socket  =  socket(AF_INET,  SOCK_RAW,  IPPROTO_IP);  //  原始套接
```

```
字，网络层
    if (INVALID_SOCKET == g_Socket) {
        show_error_msg("socket init error");
        return 0;
    }
    printf("socket init ok s=%08X\n", g_Socket);
    // 绑定
    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    addr.sin_port = 0;
    if (bind(g_Socket, (sockaddr*)&addr, sizeof(addr)) == SOCKET_ERROR)

        show_error_msg("bind error");
    }
    printf("bind ok\n");
    // 2.把网卡设置为混杂模式
    //设置网卡为混杂模式，也叫泛听模式。可以侦听经过的所有的包。
    int optval = 1;
    DWORD dwBytesRet;
    WSAIoctl(g_Socket, SIO_RCVALL, &optval, sizeof(optval),
        nullptr, 0, &dwBytesRet, nullptr, nullptr);
    /*
    WSAIoctl 参数：
    参数1：要操作的socket
    参数2：控制码
    参数3：缓冲区说明是开还是关（1开2关）
    参数7：保存操作的字节数
    */
    //3.接收数据
    char recv_buf[0x1000];
    sockaddr_in server_addr;
    int len = sizeof(server_addr);
    while (true) {
        // 接受数据
        int ret = recv(g_Socket, (char*)recv_buf, sizeof(recv_buf), 0)

        if (ret <= 0) {
            show_error_msg("[client] recv server error");
            break;
        }
        // 判断抓取到的包的类型
        ip_hdr* ip = (ip_hdr*)recv_buf;
        switch (ip->proto) {
            case IPPROTO_UDP: {
                udp_header* udp = (udp_header*)(ip + 1);
                printf("
```

```
[UDP] ip:%s srcport:%d dstprot:%d chesum:%04x bytes:%d msg:%s\n",
                    inet_ntoa(*(in_addr*)& ip->sourceIP),
                    htons(udp->src_port),
                    htons(udp->dst_prot),
                    udp->check_sum,
                    htons(udp->len),
                    (char*)(udp + 1));
                break;
            }
            case IPPROTO_TCP: {
                tcp_header* tcp = (tcp_header*)(ip + 1);
                printf("
[TCP] ip:%s srcport:%d dstprot:%d chesum:%d bytes:%d nSequNum:%d msg

                    inet_ntoa(*(in_addr*)& ip->sourceIP),
                    htons(tcp->nSourPort),
                    htons(tcp->nDestPort),
                    tcp->nCheckSum,
                    tcp->nSequNum,
                    (char*)(tcp + 1));
            }
            break;
        }
        //udp_header* udp = (udp_header*)(icmp2 + 1);
    }
    //关闭socket
    closesocket(g_Socket);
    return 0;
}
```

WireShark 的使用

不是从网络层拦截的，从物理层（网卡）拦截

Wireshark（前称Ethereal）是一个网络封包分析软件。网络封包分析软件的功能是撷取网络封包，并尽可能显示出最为详细的网络封包资料。Wireshark使用WinPCAP作为接口，直接与网卡进行数据报文交换。

socket 相关的设置

阻塞（同步）
非阻塞（异步） -- 不需要使用多线程（解决大量客户端问题）

> 相关API：
> **setsockopt**：设置一个套接字选项
> **getsockopt**：获取当前操作系统中的socket相关信息
> **ioctlsocket**：设置同步 异步
> **htons**：转换端口大小尾
> **listen**：监听

TCP 默认接受缓冲区字节大小为：0x2000 ， 4个字节

正常接收连接会阻塞 accept -- 等待连接

> 阻塞：一个线程对应一个客户端
> 非阻塞：当客户端没有数据的时候，接收下一个客户端数据，一个线程可以接受多个客户端数据。

当线程太多的时候，使用同步的效率较低，线程太多需要注意自己的系统资源是否可以支持。
使用异步，可以做到节省系统资源，常用到设计上，处理大量客户端的问题。但是客户端过大的时候，依然存在问题

服务器代码示例：

```
// socketSet.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//
#include <iostream>
#include <stdio.h>
#include <Winsock2.h>
#include <ws2tcpip.h>
#include <Mstcpip.h>
#pragma comment(lib, "Ws2_32.lib")
SOCKET g_Socket;
class CSocket {
public:
    CSocket() {
        //初始化套接字库
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            printf("
[server] WSAStartup error:%08X\n", WSAGetLastError());
        }
    }
    ~CSocket() {
        //反初始化库
        WSACleanup();
    }
}g_init;
void show_error_msg(const char* pre) {
    LPVOID lpMsgBuf;
```

```
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language

        (LPTSTR)& lpMsgBuf,
        0,
        NULL
    );
    printf("%s:%s", pre, (LPCTSTR)lpMsgBuf);
    // Free the buffer.
    LocalFree(lpMsgBuf);
}
SOCKET g_ary[100];                  // 存储socket的数组
int g_nCount = 0;
int main() {
    //1.初始化套接字(说明使用的协议)
    g_Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //原始套
接字，网络层
    if (INVALID_SOCKET == g_Socket) {
        show_error_msg("socket init error");
        return 0;
    }
    printf("socket init ok s=%08X\n", g_Socket);
    //fiddler 抓http 代理
      //阻塞(同步) 非阻塞(异步) //socket 设置
    //设置接收缓冲区的大小
    //int recv_len = 0;
    //int set_len = sizeof(recv_len);
    //getsockopt(g_Socket, SOL_SOCKET, SO_RCVBUF, (char*)& recv_len, &

    //recv_len = 0x4000;
    //setsockopt(g_Socket, SOL_SOCKET, SO_RCVBUF, (char*)&recv_len, siz

    //recv_len = 0;
    //getsockopt(g_Socket, SOL_SOCKET, SO_RCVBUF, (char*)&recv_len, &se

    ////关闭Nagle算法（关不掉）-- 产生粘包问题 TCP_NODELAY -- 不延
时
    //BOOL value = TRUE;
    //setsockopt(g_Socket, IPPROTO_TCP, TCP_NODELAY, (char*)&value, siz

    //绑定
    sockaddr_in addr;
```

```
    addr.sin_family = AF_INET;
    addr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    addr.sin_port = htons(5566);
        // 绑定
    if (bind(g_Socket, (sockaddr*)& addr, sizeof(addr)) == SOCKET_ERRO

        show_error_msg("bind error");
    }
    printf("bind ok\n");
    // 监听
    listen(g_Socket, 100);
    //设置同步异步 -- FIONBIO: 不阻塞
    u_long block = 1;
    ioctlsocket(g_Socket, FIONBIO, &block);
    while (true) {
        sockaddr_in caddr;
        int addr_len = sizeof(sockaddr_in);
        // 等待客户端连接
        SOCKET s = accept(g_Socket, (sockaddr*)& caddr, &addr_len);

        if (s == INVALID_SOCKET) {
            int ErrorCode = WSAGetLastError();
              // 连接失败，等待下一个客户端连接
            if (ErrorCode != WSAEWOULDBLOCK) {
                break;
            }
        }
        else {
            printf("accept s:%08x\n", s);
            // 连接成功，将socket保存到数组中
            g_ary[g_nCount++] = s;
        }
        //3.接收数据    延迟
        for (int i = 0; i < g_nCount; i++) {
            //轮讯设计
            char recv_buf[0x1000];                  // 存储接受到的数据
            sockaddr_in server_addr;
            int len = sizeof(server_addr);
            int ret = recv(g_ary[i], (char*)recv_buf, sizeof(recv_buf),

            if (ret > 0) {
                // 显示数据
                recv_buf[ret] = '\0';
                printf("data:%s\n", recv_buf);
            }
        }
    }
```

```
    //1.抓包
    //2.socket选项
    //3.同步、异步
    //关闭socket
    closesocket(g_Socket);
    return 0;
}
```

客户端代码示例:

```
// Client.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//
#include <stdio.h>
#include <Winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "Ws2_32.lib")
SOCKET g_Socket;
class CSocket {
public:
    CSocket() {
        //初始化套接字库
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            printf("
[server] WSAStartup error:%08X\n", WSAGetLastError());
        }
    }
    ~CSocket() {
        //反初始化库
        WSACleanup();
    }
}g_init;
void show_error_msg(const char* pre) {
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language

        (LPTSTR)& lpMsgBuf,
        0,
```

```
        NULL
    );
    printf("%s:%s", pre, (LPCTSTR)lpMsgBuf);
    // Free the buffer.
    LocalFree(lpMsgBuf);
}
int main()
{
    //1.初始化套接字(说明使用的协议)
    g_Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //TCP协议
    if (INVALID_SOCKET == g_Socket) {
        show_error_msg("[client] socket init error");
        return 0;
    }
    printf("[client] socket init ok s=%08X\n", g_Socket);
    // 2. 连接服务器（等价于三次握手）
    sockaddr_in addr; // 存储服务器的相关信息
    addr.sin_family = AF_INET;
    addr.sin_port = htons(5566);
    inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
    // 在TCP中客户端没有连接服务器，是不能向服务器发送数据的
    if (connect(g_Socket, (sockaddr*)& addr, sizeof(addr)) == SOCKET_

        show_error_msg("[client] connect server error");
        return 0;
    }
    // 连接服务器成功
    printf("[client] connect server ok\n");
    // 3. 收发数据
    char szBuff[260];
    int nRet;    // 存储向服务器发送的数据包长度
    int nLen;    // 存储发送信息的字节长度
    while (true) {
        printf("[msg]:");
        scanf_s("%s", szBuff, sizeof(szBuff));
        nLen = strlen(szBuff); // int: 65535、short: 32765
        // 发送数据
        nRet = send(g_Socket, (char*)szBuff, nLen, 0);
        if (nRet <= 0) {
            show_error_msg("[client] send server error");
            break;
        }
        printf("[client] send server ok bytes:%d\n", nRet);
    }
    //关闭socket
    closesocket(g_Socket);
```

```
    return 0;
}
```