教辅--libpcap

1. **more /usr/include/sys/types.h**

#ifndef \_SYS\_TYPES\_H

#define \_SYS\_TYPES\_H 1

#include <features.h>

\_\_BEGIN\_DECLS

#include <bits/types.h>

#ifdef \_\_USE\_BSD

# ifndef \_\_u\_char\_defined

typedef \_\_u\_char u\_char;

typedef \_\_u\_short u\_short;

typedef \_\_u\_int u\_int;

typedef \_\_u\_long u\_long;

# define \_\_u\_char\_defined

# endif

#endif

typedef \_\_loff\_t loff\_t;

#ifndef \_\_ino\_t\_defined

# ifndef \_\_USE\_FILE\_OFFSET64

typedef \_\_ino\_t ino\_t;

# else

typedef \_\_ino64\_t ino\_t;

# endif

# define \_\_ino\_t\_defined

#endif

#if defined \_\_USE\_LARGEFILE64 && !defined \_\_ino64\_t\_defined

typedef \_\_ino64\_t ino64\_t;

# define \_\_ino64\_t\_defined

#endif

#ifndef \_\_gid\_t\_defined

typedef \_\_gid\_t gid\_t;

# define \_\_gid\_t\_defined

#endif

#ifndef \_\_mode\_t\_defined

typedef \_\_mode\_t mode\_t;

# define \_\_mode\_t\_defined

#endif

#ifndef \_\_uid\_t\_defined

typedef \_\_uid\_t uid\_t;

# define \_\_uid\_t\_defined

#endif

#ifndef \_\_off\_t\_defined

# ifndef \_\_USE\_FILE\_OFFSET64

typedef \_\_off\_t off\_t;

# else

typedef \_\_off64\_t off\_t;

# endif

# define \_\_off\_t\_defined

#endif

#ifndef \_\_pid\_t\_defined

typedef \_\_pid\_t pid\_t;

# define \_\_pid\_t\_defined

#endif

#ifndef \_\_ssize\_t\_defined

typedef \_\_ssize\_t ssize\_t;

# define \_\_ssize\_t\_defined

#endif

**#ifdef \_\_USE\_MISC**

**/\* Old compatibility names for C types. \*/**

**typedef unsigned long int ulong;**

**typedef unsigned short int ushort;**

**typedef unsigned int uint;**

**#endif**

/\* These types are defined by the ISO C99 header <inttypes.h>. \*/

# ifndef \_\_int8\_t\_defined

**# define \_\_int8\_t\_defined**

**typedef char int8\_t;**

**typedef short int int16\_t;**

**typedef int int32\_t;**

# if \_\_WORDSIZE == 64

typedef long int int64\_t;

# elif \_\_GLIBC\_HAVE\_LONG\_LONG

\_\_extension\_\_ typedef long long int int64\_t;

# endif

# endif

/\* But these were defined by ISO C without the first `\_'. \*/

**typedef unsigned char u\_int8\_t;**

**typedef unsigned short int u\_int16\_t;**

**typedef unsigned int u\_int32\_t;**

**# if \_\_WORDSIZE == 64**

**typedef unsigned long int u\_int64\_t;**

**# elif \_\_GLIBC\_HAVE\_LONG\_LONG**

\_\_extension\_\_ typedef unsigned long long int u\_int64\_t;

# endif

typedef int register\_t;

#else

#ifdef \_\_USE\_BSD

/\* In BSD <sys/types.h> is expected to define BYTE\_ORDER. \*/

# include <endian.h>

/\* It also defines `fd\_set' and the FD\_\* macros for `select'. \*/

# include <sys/select.h>

#endif /\* Use BSD. \*/

/\* Now add the thread types. \*/

#if defined \_\_USE\_POSIX199506 || defined \_\_USE\_UNIX98

# include <bits/pthreadtypes.h>

#endif

\_\_END\_DECLS

#endif /\* sys/types.h \*/

**2. pcap.h头文件**

#ifndef lib\_pcap\_pcap\_h

#define lib\_pcap\_pcap\_h

#if defined(WIN32)

#include <pcap-stdinc.h>

#elif defined(MSDOS)

#include <sys/types.h>

#include <sys/socket.h> /\* u\_int, u\_char etc. \*/

#else /\* UN\*X \*/

**#include <sys/types.h>**

**#include <sys/time.h>**

#endif /\* WIN32/MSDOS/UN\*X \*/

#ifndef PCAP\_DONT\_INCLUDE\_PCAP\_BPF\_H

#include <pcap/bpf.h>

#endif

#include <stdio.h>

**#define PCAP\_ERRBUF\_SIZE 256**

/\* \* Compatibility for systems that have a bpf.h that predates the bpf typedefs for 64-bit support. \*/

#if BPF\_RELEASE - 0 < 199406

**typedef int bpf\_int32;**

**typedef u\_int bpf\_u\_int32;**

#endif

**typedef struct pcap pcap\_t;**

**typedef struct pcap\_dumper pcap\_dumper\_t;**

**typedef struct pcap\_if pcap\_if\_t;**

**typedef struct pcap\_addr pcap\_addr\_t;**

**struct pcap\_file\_header {**

**bpf\_u\_int32 magic;**

**u\_short version\_major;**

**u\_short version\_minor;**

**bpf\_int32 thiszone; /\* gmt to local correction \*/**

**bpf\_u\_int32 sigfigs; /\* accuracy of timestamps \*/**

**bpf\_u\_int32 snaplen; /\* max length saved portion of each pkt \*/**

**bpf\_u\_int32 linktype; /\* data link type (LINKTYPE\_\*) \*/**

**};**

typedef enum {

PCAP\_D\_INOUT = 0,

PCAP\_D\_IN,

PCAP\_D\_OUT

} pcap\_direction\_t;

/\* \* Generic per-packet information, as supplied by libpcap. \*/

**struct pcap\_pkthdr {**

**struct timeval ts; /\* time stamp \*/**

**bpf\_u\_int32 caplen; /\* length of portion present \*/**

**bpf\_u\_int32 len; /\* length this packet (off wire) \*/**

**};**

/\* \* As returned by the pcap\_stats() \*/

**struct pcap\_stat {**

**u\_int ps\_recv; /\* number of packets received \*/**

**u\_int ps\_drop; /\* number of packets dropped \*/**

**u\_int ps\_ifdrop; /\* drops by interface -- only supported on some platforms \*/**

#ifdef WIN32

u\_int bs\_capt; /\* number of packets that reach the application \*/

#endif /\* WIN32 \*/

};

/\* \* Item in a list of interfaces. \*/

**struct pcap\_if {**

**struct pcap\_if \*next;**

**char \*name; /\* name to hand to "pcap\_open\_live()" \*/**

**char \*description; /\* textual description of interface, or NULL \*/**

**struct pcap\_addr \*addresses;**

**bpf\_u\_int32 flags; /\* PCAP\_IF\_ interface flags \*/**

**};**

**#define PCAP\_IF\_LOOPBACK 0x00000001 /\* interface is loopback \*/**

/\* Representation of an interface address. \*/

**struct pcap\_addr {**

**struct pcap\_addr \*next;**

**struct sockaddr \*addr; /\* address \*/**

**struct sockaddr \*netmask; /\* netmask for that address \*/**

**struct sockaddr \*broadaddr; /\* broadcast address for that address \*/**

**struct sockaddr \*dstaddr; /\* P2P destination address for that address \*/**

**};**

typedef void (\*pcap\_handler)(u\_char \*, const struct pcap\_pkthdr \*, const u\_char \*);

/\* \* Error codes for the pcap API. These will all be negative, so you can check for the success or

\* failure of a call that returns these codes by checking for a negative value. \*/

**#define PCAP\_ERROR -1 /\* generic error code \*/**

**#define PCAP\_ERROR\_BREAK -2 /\* loop terminated by pcap\_breakloop \*/**

**#define PCAP\_ERROR\_NOT\_ACTIVATED -3 /\* the capture needs to be activated \*/**

**#define PCAP\_ERROR\_ACTIVATED -4 /\* the operation can't be performed on already activated captures \*/**

**#define PCAP\_ERROR\_NO\_SUCH\_DEVICE -5 /\* no such device exists \*/**

**#define PCAP\_ERROR\_RFMON\_NOTSUP -6 /\* this device doesn't support rfmon (monitor) mode \*/**

**#define PCAP\_ERROR\_NOT\_RFMON -7 /\* operation supported only in monitor mode \*/**

**#define PCAP\_ERROR\_PERM\_DENIED -8 /\* no permission to open the device \*/**

**#define PCAP\_ERROR\_IFACE\_NOT\_UP -9 /\* interface isn't up \*/**

**#define PCAP\_ERROR\_CANTSET\_TSTAMP\_TYPE -10 /\* this device doesn't support setting the time stamp type \*/**

**#define PCAP\_ERROR\_PROMISC\_PERM\_DENIED -11 /\* you don't have permission to capture in promiscuous mode \*/**

**#define PCAP\_ERROR\_TSTAMP\_PRECISION\_NOTSUP -12 /\* the requested time stamp precision is not supported \*/**

/\* \* Warning codes for the pcap API. These will all be positive and non-zero, so they won't look like errors. \*/

**#define PCAP\_WARNING 1 /\* generic warning code \*/**

**#define PCAP\_WARNING\_PROMISC\_NOTSUP 2 /\* this device doesn't support promiscuous mode \*/**

**#define PCAP\_WARNING\_TSTAMP\_TYPE\_NOTSUP 3 /\* the requested time stamp type is not supported \*/**

/\*\* Value to pass to pcap\_compile() as the netmask if you don't know what the netmask is. \*/

#define PCAP\_NETMASK\_UNKNOWN 0xffffffff

char \*pcap\_lookupdev(char \*);

int pcap\_lookupnet(const char \*, bpf\_u\_int32 \*, bpf\_u\_int32 \*, char \*);

pcap\_t \*pcap\_create(const char \*, char \*);

int pcap\_set\_snaplen(pcap\_t \*, int);

int pcap\_set\_promisc(pcap\_t \*, int);

int pcap\_set\_timeout(pcap\_t \*, int);

.......................................................................

PCAP库所有函数原型的定义

.......................................................................

#ifndef \_\_NetBSD\_\_

u\_int bpf\_filter(const struct bpf\_insn \*, const u\_char \*, u\_int, u\_int);

#endif

int bpf\_validate(const struct bpf\_insn \*f, int len);

char \*bpf\_image(const struct bpf\_insn \*, int);

void bpf\_dump(const struct bpf\_program \*, int);

#if defined(WIN32)

/\* \* Win32 definitions \*/

int pcap\_setbuff(pcap\_t \*p, int dim);

int pcap\_setmode(pcap\_t \*p, int mode);

int pcap\_setmintocopy(pcap\_t \*p, int size);

Adapter \*pcap\_get\_adapter(pcap\_t \*p);

#ifdef WPCAP

/\* Include file with the wpcap-specific extensions \*/

#include <Win32-Extensions.h>

#endif /\* WPCAP \*/

#define MODE\_CAPT 0

#define MODE\_STAT 1

#define MODE\_MON 2

#else /\* UN\*X \*/

/\* \* UN\*X definitions \*/

int pcap\_get\_selectable\_fd(pcap\_t \*);

#endif /\* lib\_pcap\_pcap\_h \*/

1. 程序simplesniffer.c

/\* To compile: gcc simplesniffer.c -o simplesniffer -lpcap \* Run as root! \*/

#include <pcap.h>

#include <string.h>

#include <stdlib.h>

#define MAXBYTES2CAPTURE 2048

/\* processPacket(): Callback function called by pcap\_loop() everytime a packet \*/

/\* arrives to the network card. This function prints the captured raw data in hexadecimal. \*/

void processPacket(u\_char \*arg, const struct pcap\_pkthdr\* pkthdr, const u\_char \* packet){

int i=0, \*counter = (int \*)arg;

printf("Packet Count: %d\n", ++(\*counter));

printf("Received Packet Size: %d\n", pkthdr->len);

printf("Payload:\n");

for (i=0; i<pkthdr->len; i++){

if ( isprint(packet[i]) ) /\* If it is a printable character, print it \*/

printf("%c ", packet[i]);

else

printf(". ");

if( (i%16 == 0 && i!=0) || i==pkthdr->len-1 )

printf("\n");

}

return;

}

/\* main(): Main function. Opens network interface and calls pcap\_loop() \*/

int main(int argc, char \*argv[] ){

int i=0, count=0;

pcap\_t \*descr = NULL;

char errbuf[PCAP\_ERRBUF\_SIZE], \*device=NULL;

memset(errbuf,0,PCAP\_ERRBUF\_SIZE);

if( argc > 1){ /\* If user supplied interface name, use it. \*/

device = argv[1];

}

else{ /\* Get the name of the first device suitable for capture \*/

if ( (device = pcap\_lookupdev(errbuf)) == NULL){

fprintf(stderr, "ERROR: %s\n", errbuf);

exit(1);

}

}

printf("Opening device %s\n", device);

/\* Open device in promiscuous mode \*/

if ( (descr = pcap\_open\_live(device, MAXBYTES2CAPTURE, 1, 512, errbuf)) == NULL){

fprintf(stderr, "ERROR: %s\n", errbuf);

exit(1);

}

/\* Loop forever & call processPacket() for every received packet\*/

if ( pcap\_loop(descr, 10, processPacket, (u\_char \*)&count) == -1){

fprintf(stderr, "ERROR: %s\n", pcap\_geterr(descr) );

exit(1);

}

return 0;

}

2. packet中的data部分



/\* Ethernet addresses are 6 bytes \*/

#define ETHER\_ADDR\_LEN 6

/\* Ethernet header \*/

struct sniff\_ethernet {

u\_char ether\_dhost[ETHER\_ADDR\_LEN]; /\* Destination host address \*/

u\_char ether\_shost[ETHER\_ADDR\_LEN]; /\* Source host address \*/



u\_short ether\_type; /\* IP? ARP? RARP? etc \*/

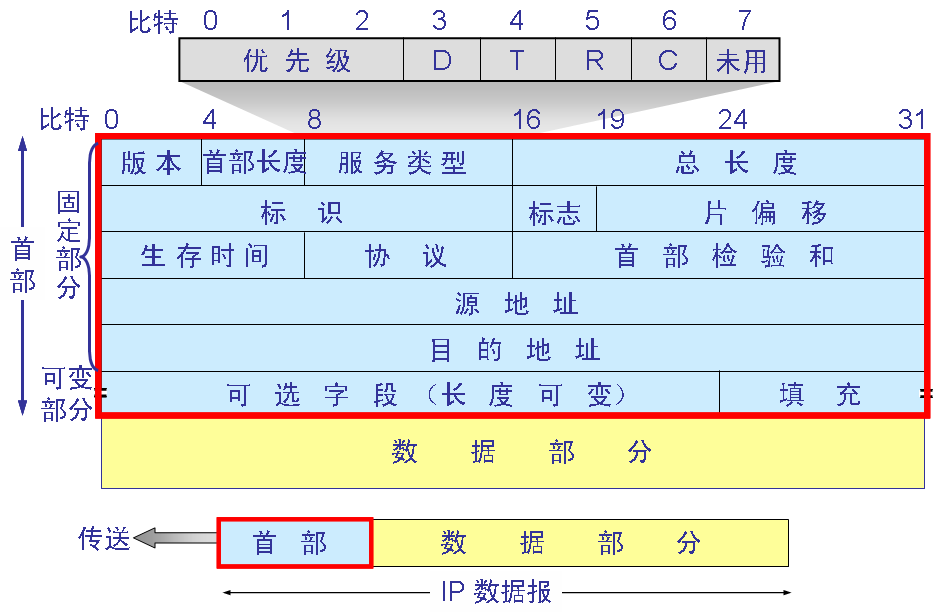
};

/\* IP header \*/

struct sniff\_ip {

u\_char ip\_vhl; /\* version << 4 | header length >> 2 \*/

u\_char ip\_tos; /\* type of service \*/



u\_short ip\_len; /\* total length \*/

u\_short ip\_id; /\* identification \*/

u\_short ip\_off; /\* fragment offset field \*/

#define IP\_RF 0x8000 /\* reserved fragment flag \*/

#define IP\_DF 0x4000 /\* dont fragment flag \*/

#define IP\_MF 0x2000 /\* more fragments flag \*/

#define IP\_OFFMASK 0x1fff /\* mask for fragmenting bits \*/

u\_char ip\_ttl; /\* time to live \*/

u\_char ip\_p; /\* protocol \*/

u\_short ip\_sum; /\* checksum \*/

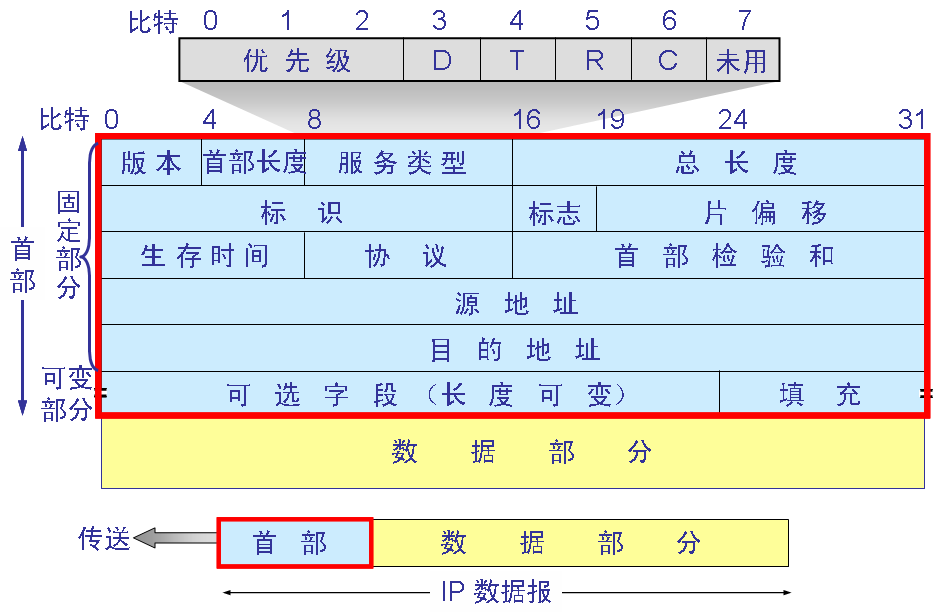
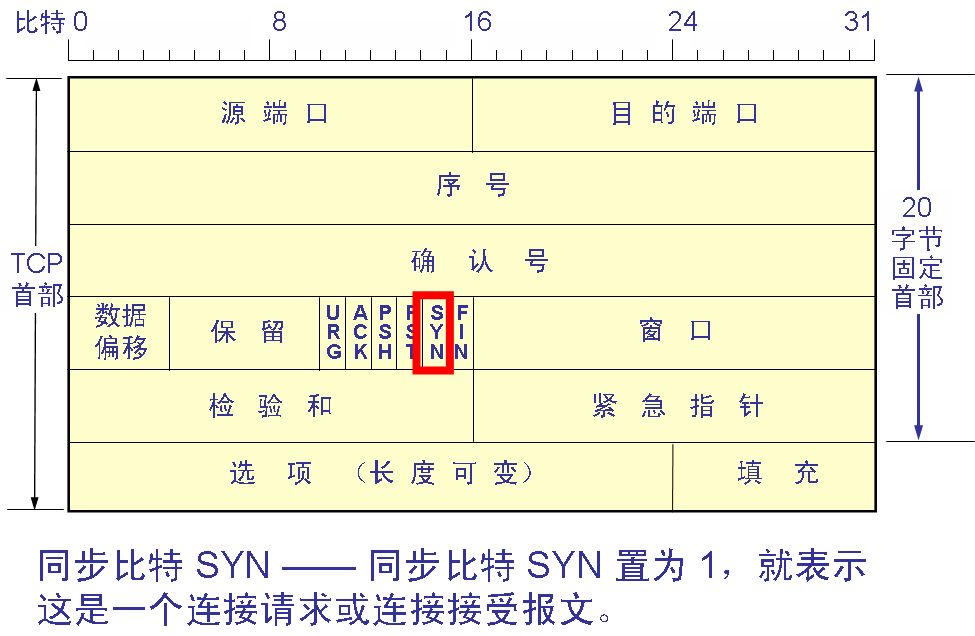
struct in\_addr ip\_src,ip\_dst; /\* source and dest address \*/

};

#define IP\_HL(ip) (((ip)->ip\_vhl) & 0x0f)

#define IP\_V(ip) (((ip)->ip\_vhl) >> 4)

/\* TCP header \*/



typedef u\_int tcp\_seq;

struct sniff\_tcp {

u\_short th\_sport; /\* source port \*/

u\_short th\_dport; /\* destination port \*/

tcp\_seq th\_seq; /\* sequence number \*/

tcp\_seq th\_ack; /\* acknowledgement number \*/

u\_char th\_offx2; /\* data offset, rsvd \*/

#define TH\_OFF(th) (((th)->th\_offx2 & 0xf0) >> 4)

u\_char th\_flags;

#define TH\_FIN 0x01

#define TH\_SYN 0x02

#define TH\_RST 0x04

#define TH\_PUSH 0x08

#define TH\_ACK 0x10

#define TH\_URG 0x20

#define TH\_ECE 0x40

#define TH\_CWR 0x80

#define TH\_FLAGS (TH\_FIN|TH\_SYN|TH\_RST|TH\_ACK|TH\_URG|TH\_ECE|TH\_CWR)

u\_short th\_win; /\* window \*/

#include <sys/types.h>/usr/include/sys/types.h

#include <netinet/in.h> /usr/include/netinet/in.h

typedef uint32\_t in\_addr\_t;

struct in\_addr

{

in\_addr\_t s\_addr;

};

u\_short th\_sum; /\* checksum \*/

u\_short th\_urp; /\* urgent pointer \*/

｝；

3.函数指针的使用

1）、简单函数指针的应用

格式： 返回类型（\*函数名）（参数表）

例：char (\*pFun)(int); 定义指针变量pFun, 它是一个指向某种函数的指针，函数参数是一个int型

返回值是char类型，但尚无法使用这个指针，因为还未对它进行赋值

char glFun(int a) {return ;} 定义了一个函数glFun()，它是一个以 int为参数，返回值是 char类型

从指针上理解函数：函数的函数名实际上就是一个指针，

函数名指向该函数的代码在内存中的首地址

void main()

{

pFun=glFun; 将函数glFun的地址赋值给变量pFun

(\*pFun)(2); (\*pFun)是取pFun所指向地址的内容，也就是取出函数glFun()的内容，

给定参数为2

}

2）. 使用typedef定义

原因：更加直观、方便

格式：typedef 返回类型 (\*新类型)(参数表)

例：typedef char (\*PTRFUN)(int); 定义了一种PTRFUN函数，并定义该类型为指向某种函数的指针

该函数以一个int为参数并返回char类型。后面的程序就可以使用

PTRFUN类型了

PTRFUN pFun; 使用新类型定义了变量pFun，此时就可以像使用简单函数指针一样使用

此变量

char glfun(int a) {return;}

void main()

{

pFun=glFun;

(\*pFun)(2);

}