

[键入公司名称]

网络编程技术

计算机网络课程设计

作者：周芳 张爽

目 录

1	Winpcap 网络编程技术	1
1.1	Winpcap 资料下载	1
1.2	Winpcap 开发环境配置	1
1.2.1	VC6.0 环境配置	1
1.2.2	VS2010 环境配置	2
2	Winpcap 开源库	4
2.1	Winpcap 的开发流程	4
2.1.1	Winpcap 概述	4
2.1.2	winpcap 的开发流程	6
2.2	Winpcap 常用基本函数	7
2.2.1	获取设备列表	7
2.2.2	打开/关闭选择的适配器	8
2.2.3	捕获数据包	10
2.2.4	过滤数据包	11
2.2.5	发送数据包	14
2.3	Winpcap 分析实例	19
2.3.1	捕获解析 UDP 数据包实例	19
2.3.2	打印通过适配器的数据包实例	24
3	WinSocket 网络编程技术	27
3.1	WinSocket 概述	27
3.2	Winsocket 开发流程	27
3.3	Winsocket 基本函数	28
3.4	Winsockets 编程实例	30

Winpcap 学习资料

Winpcap 是网络低层开发的重要工具，是在 windows 平台上访问网络数据链路层的开源库，允许应用程序避开网络协议，直接处理数据包。socket 是应用广泛的网络接口，但是 socket 经过了操作系统处理（协议处理），提供的函数接口是剥离网络协议的网络数据；而 winpcap 是直接对原始数据包进行处理，即用户自己对要传输的网络数据按照协议的首部格式进行封装，用户自己完成协议需要封装的内容，操作的是原始数据包。

下面详细介绍 winpcap 的安装和库函数的具体使用。

1 Winpcap 网络编程技术

1.1 Winpcap 资料下载

使用 Winpcap 开源库必须下载并安装 WinPcap Driver、DLL 和 wpdpack (developer's pack)。Winpcap 的官方网站上有这两个软件的最新版本。其下载地址是：<http://www.winpcap.org/archive/>

步骤 1：下载并安装，安装后重启机器。Winpcap 驱动的安装包 (Winpcap_4_1.exe)；

步骤 2：下载程序员开发包(WpdPack_4_1.zip)，解压后会看到其中包含了 docs、Include、lib、Examples 等文件夹。

winpcap 中文使用手册地址：<http://www.ferrisxu.com/WinPcap/html/index.html>

Winpcap 功能强大，效率高，使用方便，适应很多平台，通常高校的教学语言都是基于 VC++6.0 版本，有的使用 VS2010 版本，另外，由于 JAVA 对网络编程的良好支持，也有很多编程者使用 JAVA 环境。不同应用程序的 winpcap 的配置如下所述。

1.2 Winpcap 开发环境配置

1.2.1 VC6.0 环境配置

VC++6.0 是初学 C 类语言的教学环境，其配置使用 Winpcap 的步骤如下：

步骤 1：在 VC 中设定 Include 目录。打开 VC 菜单，Tools->Option->Directories，

在 include files 中添加.....\wpdpack\Include 目录（安装 wpdpack 中得到的）；

步骤 2：在 VC 中设定 Library 目录，在 Library files 中添加.....\wpdpack\Lib 目录；

步骤 3：Project->settings->Link，在 Object/library modules 中加上 wpcap.lib。

1.2.2 VS2010 环境配置

VS2010 是目前项目开发人员习惯使用的开发环境，在该环境中，配置 winpcap 非常方便。

步骤 1：在项目选项卡中点击属性，弹出以下界面：

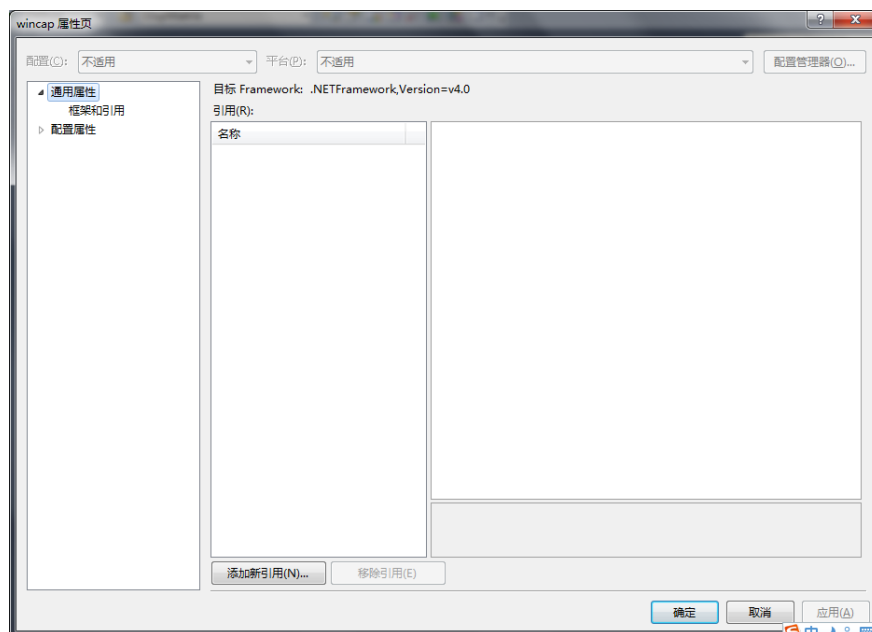


图 1.1 VS2010 中属性配置选项卡

步骤 2：点击配置属性中的 VC++ 目录选项，在包含目录和库目录中添加相应的 Include 的文件夹和 Lib 文件夹

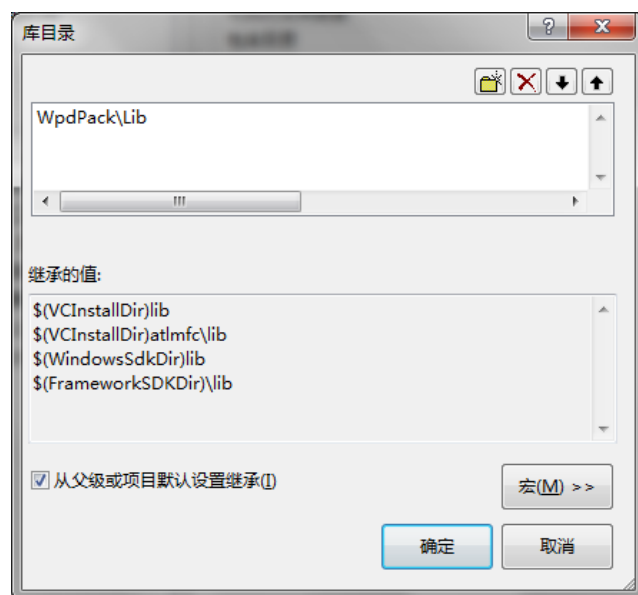
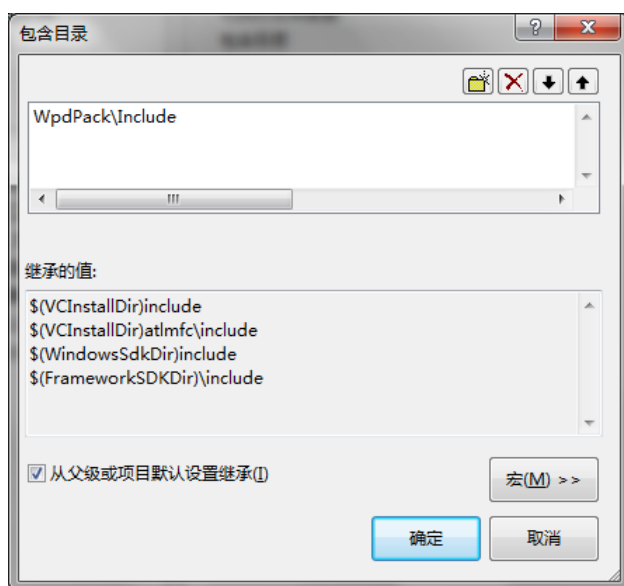
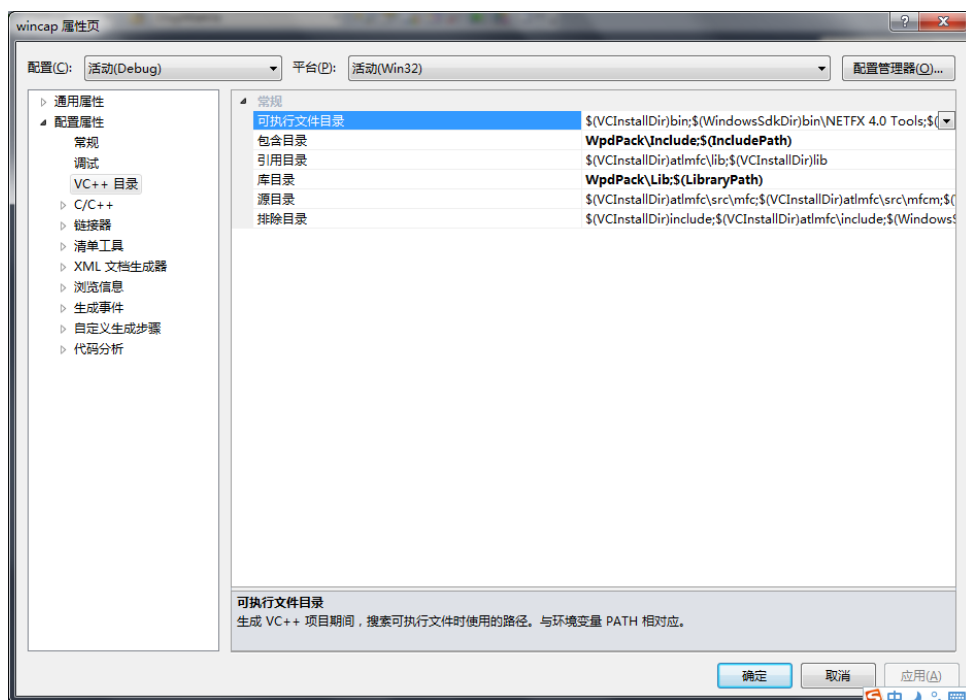


图 1.2 VS2010 中包含和库目录配置

步骤3: 代码中添加 `#pragma comment(lib, "wpcap.lib")` 即可。

(如需添加其他.lib,可如上述代码一样, 动态添加库)

例如:

```
#include "stdafx.h"
#include "pcap.h"
#include "inc.h"
#include "windows.h"

#pragma comment(lib, "wpcap.lib")
#pragma comment(lib, "ws2_32")
```

2 Winpcap 开源库

winpcap 是 win32 平台下进行网络捕获和网络分析的开源库。它避开了操作系统对网络数据的隐藏，直接对数据链路层的数据包进行处理，也就是原始的网络数据进行操作，可以方便的进行网络数据的封装和处理。对已经学过计算机网络网络的组成原理的编程初学人员来讲，使用 winpcap 开发网络应用程序，既能学习网络编程的基本过程，又能在编程过程中加深对网络协议栈的深入理解，是培养计算机网络专业人才的很好的途径之一。

winpcap 提供的基本功能有：

- 捕获经由主机的数据包；
- 根据应用程序提供的规则（程序员自己定义）过滤数据包；
- 发送原始数据包到网络上；
- 统计和手机网络流量信息。

2.1 Winpcap 的开发流程

2.1.1 Winpcap 概述

Winpcap 包括三个组成部分：

- 第一个模块：内核级的包过滤驱动程序 NPF(Netgroup Packet Filter)
- 第二个模块：低级动态链接库 packet.dll，在 Win32 平台上提供了与 NPF 的一个通用接口
- 第三个模块：用户级的 wpcap.dll，调用 packet.dll 提供的函数提供更高级的功能

这三个部分的关系如图所示：

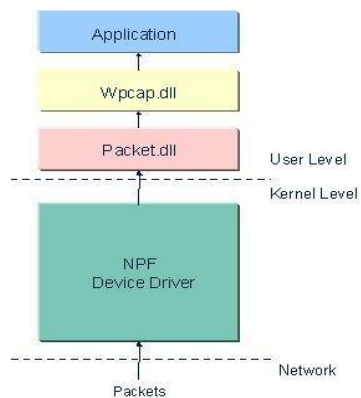


图 2.1 Winpcap 组成

第一个模块是网络数据包过滤器（Netgroup Filter, NPF），它是 Winpcap 的核心部分，负责处理网络上传输的数据包，并且对用户级提供捕获、发送和分析等功能。

第二个模块 packet.dll 和第三个模块 wpcap.dll 提供了基于用户级的接口函数库。packet.dll 提供了一个底层 API，这些 API 可以直接用来访问内核；wpcap.dll 提供一组更加友好、功能更强大的高层函数库。

用户编程实现网络捕获功能时，可以直接使用第一个模块的函数，也可以直接使用第二个模块的函数，当然也可以使用第三个模块的函数。应用程序和上述三个模块之间的关系如下图所示。

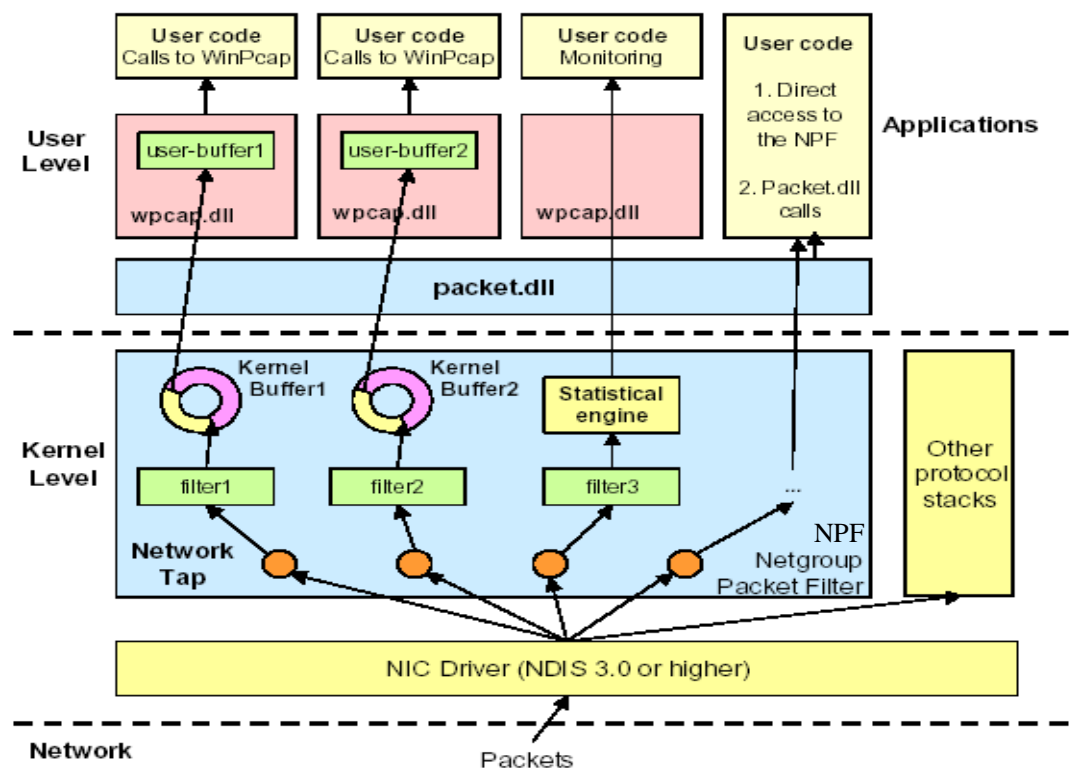
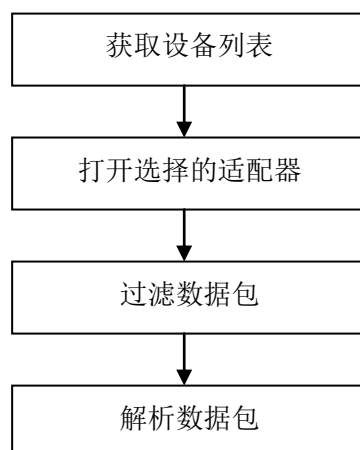


图 2.2 Winpcap 模块关系图

2.1.2 winpcap 的开发流程

winpcap 的开发流程大致如下：



2.2 Winpcap 常用基本函数

2.2.1 获取设备列表

winpcap 中网络适配器信息用一个结构体 `pcap_if_t` 描述，其结构定义如下：

```
typedef struct pcap_if_t;
struct pcap_if_t
{
    struct pcap_if_t *next;           //如果不为空，则指向下一个元素
    char *name;                       //设备名称
    char *description;                //描述设备
    struct pcap_addr *addresses;      //接口地址列表
    bpf_u_int32 flags;                //
};
```

每个 `pcap_if_t` 结构包含了一个 `pcap_addr` 结构的列表：

- ❖ 该接口的地址列表
- ❖ 网络掩码的列表（每个网络掩码对应地址列表中的一项）
- ❖ 广播地址的列表（每个广播地址对应地址列表中的一项）
- ❖ 目标地址的列表（每个目标地址对应地址列表中的一项）

`pcap_addr` 结构定义如下：

```
typedef struct pcap_addr pcap_addr_t;
struct pcap_addr
{
    struct pcap_addr *next;           //如果不为空，则指向下一个元素
    struct sockaddr *addr;            //接口 IP 地址
    struct sockaddr *netmask;         //接口网络掩码
    struct sockaddr *broadaddr;       //接口广播地址
    struct sockaddr *dstaddr;         //接口 P2P 目的地址
};
```

● `pcap_findalldevs()` 函数

Libpcap 提供 `pcap_findalldevs()` 函数完成查找网络适配器功能，该函数返回本地主机上安装的所有适配器列表。函数返回一个相连的 `pcap_if` 结构的列表，该列表的每一项包含关于适配器的复杂的信息。

`pcap_findalldevs()` 函数原型如下：

```
int pcap_findalldevs(pcap_if_t  **alldevs,          //指向列表的第一个元素，列表
                                                            元素每个都是 pcap_if_t 类型，如果没有
                                                            已连接并打开的适配器，则为 NULL

                    char *errbuf)                    //存储错误信息

函数调用成功返回 0，失败返回-1。
```

例程：

```
pcap_if_t  *alldevs, *d;
int  i=0;
char errbuf[PCAP_ERRBUF_SIZE];
if (pcap_findalldevs(&alldevs, errbuf) == -1)
{  fprintf(stderr, "Error in pcap_findalldevs: %s\n", errbuf);
   exit(1);
}
for(d=alldevs; d; d=d->next)
{  printf("%d. %s", ++i, d->name);                      /* Print the list
*/
   if (d->description)
       printf(" (%s)\n", d->description);
   else
       printf(" (No description available)\n");
}
if (i==0)
{  printf("\nNo interfaces found! \n");
   return ;
}
pcap_freealldevs(alldevs);
```

2.2.2 打开/关闭选择的适配器

● 打开适配器 pcap_open_live ()

通过 pcap_findalldevs()找到本地主机上安装的所有网络适配器，选择其中的一个进行网络编程必须打开它，其使用的函数是 pcap_open_live ()，其功能是打开本地主机上的网络适配卡，函数原型描述如下：

```
pcap_t  * pcap_open_live ( const char  * device,
                            int  snaplen,
                            int  promisc,
                            int  to_ms,
                            char  * ebuf)
```

其中的参数说明如下：

const char *device: 所选择网络适配器的设备标识（字符串）；
int snaplen: 进行捕获的数据包长度限制；
int promisc: 是否以混杂模式进行捕获；
int to_ms: 捕获数据包能容忍的超时时间；
char *ebuf: 出错信息缓存；

函数 pcap_open_live（）的返回值是 pcap_t 类型的指针，pcap_t 结构体对用户透明，提供了对一个已打开的适配器实例的描述。windows 平台上 pcap_t 的主要成员有（只列举几个重要成员）：

```
typedef struct pcap pcap_t;
struct pcap
{
    ADAPTER *adapter;
    LPPACKET Packet;
    int linktype;           //数据链路层类型
    int linktype_ext;       // linktype 成员扩展信息
    int offset;             //时区偏移
    int activated;          //捕获准备好否
    struct pcap_sf sf;
    struct pcap_md md;
    struct pcap_opt opt;
    ...
};
```

例如：

```
pcap_t *adhandle = pcap_open_live(d->name,    //适配器名字
                                   65535,      //捕获包最大字节数
                                   1,           //混杂模式
                                   1000,        //读取超时时间
                                   errbuf);    //错误信息保存
```

● 关闭适配器

打开某个网络适配器，使用结束后必须关闭它。关闭适配器的函数是 pcap_close（），其函数原型描述如下：

```
void pcap_close ( pcap_t *p);
```

该函数的参数是 pcap_t 类型的指针。函数执行结果释放掉 p 指向的网络适配器。

2.2.3 捕获数据包

捕获数据包通常有两种方式：一种是直接捕获；另一种是回调模式。

- 直接捕获数据包方式

使用 `int pcap_next_ex()` 直接捕获数据包，其函数原型描述如下：

<code>int pcap_next_ex (pcap_t * p,</code>	<code>//适配器名称</code>
<code>struct pcap_pkthdr ** pkt_header,</code>	<code>//捕获的数据包的首部指针</code>
<code>const u_char ** pkt_data)</code>	<code>//捕获的数据包的数据</code>

函数的返回值是整型，各取值的含义如下：

- 1: 数据包读取成功；
- 0: 如果超时时间到，则 `pkt_header` 和 `pkt_data` 都不指向有用的数据包；
- -1: 出现错误；
- -2: 离线捕获（文件操作）遇到文件尾部的 EOF。

该函数从适配器或脱机文件读取一个数据包。用于接收下一个可用的数据包。`pcap_next_ex()` 目前只在 Win32 下可用，因为它不是属 libpcap 原始的 API。这意味着含有这个函数的代码将不能被移植到 Unix 上。

其中结构体 `pcap_pkthdr` 定义如下：

```
struct pcap_pkthdr {  
    struct timeval  ts;           //时间戳  
    bpf_u_int32    caplen;       //当前分组长度  
    bpf_u_int32    len;         //数据包的长度  
}
```

例程：

<pre>while ((res = pcap_next_ex(adhandle, &header, &pkt_data)) >= 0) { if (res == 0) { continue; /* Timeout elapsed */ } /* convert the timestamp to readable format */ ltime = localtime(&header->ts.tv_sec); strftime(timestr, sizeof(timestr), "%H:%M:%S", ltime); printf("%s, %.6d len:%d\n", timestr, header->ts.tv_usec, header->len); } if (res == -1) { printf("Error reading the packets: %s\n", pcap_geterr(adhandle)); return -1; }</pre>
--

- 回调方式捕获数据包

使用 `int pcap_loop ()` 直接捕获数据包，其函数原型描述如下：

```
int pcap_loop ( pcap_t * p,
                int cnt,
                pcap_handler callback,
                u_char * user )
```

其中 `pcap_handler` 函数原型定义如下：

```
typedef void (* pcap_handler)
            ( u_char *user,
              const struct pcap_pkthdr *pkt_header,
              const u_char *pkt_data);
```

例程：

```
void  pcap_handler (u_char* user,  const struct pcap_pkthdr* pkt_header,
                   const u_char* pkt_data);
const  struct  pcap_pkthdr  *header, const  u_char  *pkt_data);

int main( )
{   pcap_t  *adhandle;

    pcap_loop( adhanlde, 0, packet_handler, NULL);

    return  0;
}
void  pcap_handler (u_char* user,  const struct pcap_pkthdr* pkt_header,
                   const u_char* pkt_data);
{//捕获的数据包通过该函数回传给应用程序
}
```

2.2.4 过滤数据包

Winpcap 提供的常用的过滤数据包的函数：一个是 `pcap_compile()`，另一个是 `pcap_setfilter()`。

使用 `pcap_compile()` 使用捕获的数据包的过滤。该函数编译一个数据包过滤器，将一个高级的、布尔形式表示的字符串转换成低级的、二进制过滤语句，能够被过滤虚拟机执行。`pcap_setfilter()` 在核心驱动中将过滤器和捕获过程结合在一起。从这一时刻起，所有网络的数据包都要经过过滤，通过过滤的数据包将被传入应用程序。

```

int pcap_compile ( pcap_t * p,
                  struct bpf_program * fp,
                  char * str,                //过滤表达式
                  int optimize,              //控制是否对最终生成的字节码优化
                  bpf_u_int32 netmask )      //需捕获数据包的 IPV4 掩码

int pcap_setfilter ( pcap_t * p,            //
                   struct bpf_program * fp ) //通常是 pcap_compile ( ) 返回的结果

```

其中结构体 bpf_program 定义如下:

```

struct bpf_program{ u_int bf_len;           //BPF 中谓词判断指令的数目
                   struct bpf_insn *bf_insns; //指向第一个谓词判断指令
};

```

函数 pcap_compile() 执行成功返回 0, 否则返回-1。可以调用 pcap_geterr 函数显示所发生的错误信息。

pcap_setfilter() 被调用时, 该过滤器被应用到来自网络的所有数据包上, 所有符合过滤要求的数据包将会被存储到内核缓冲区中。pcap_setfilter() 函数执行成功返回 0, 失败返回-1, 调用 pcap_geterr 函数显示所发生的错误信息。

例程:

```

char packet_filter[] = "ip and udp";
struct bpf_program fcode;
/* 获取接口地址的掩码, 如果没有掩码, 认为该接口属于一个 C 类网络 */
if (d->addresses != NULL)
    netmask=((struct sockaddr_in *)
            (d->addresses->netmask))->sin_addr.S_un.S_addr;
else netmask=0xffffffff;
if (pcap_compile(adhandle, &fcode, packet_filter, 1, netmask) < 0 )
{
    fprintf(stderr, "\nUnable to compile the filter. Check the syntax.\n");
    pcap_freealldevs(alldevs);
    return -1;
}
if (pcap_setfilter(adhandle, &fcode) < 0)
{
    fprintf(stderr, "\nError setting the filter.\n");
    pcap_freealldevs(alldevs);
    return -1;
}

```

pcap_compile() 中的过滤表达式由一个或多个原语组成。原语通常由一个 id(名称或者数字)和在它前面的一个或几个修饰符组成。有 3 种不同的修饰符:

类型 指明 id 名称或者数字指的是哪种类型

可能是 host, net 和 port。例如 “host foo”、“net 128.3”、“port 20”。如果没有类型修饰符, 缺省为 host。

方向 指明向和/或 从 id 传输等方向的修饰符

可能的方向有 src、dst、src or dst 和 src and dst。例如“src foo”、“dst net 128.3”、“src or dst port ftp-data”。如果缺省为 src or dst。

协议 指明符合特定协议的修饰符

目前的协议包括 ether、fddi、ip、ip6、arp、rarp、tcp 和 udp 等。例如“ether src foo”、“arp net 128.3”。

如果没有协议修饰符, 则表示声明类型的所有协议。

例如“src foo”表示“(ip or arp or rarp) src foo”

“port 53”表示“(tcp or udp) port 53”。

编译并设置过滤器过滤函数的实例如下:

```
if (d->addresses != NULL)
    /* 获取接口第一个地址的掩码 */
    netmask=((struct sockaddr_in *)(d->addresses->netmask))->sin_addr.S_un.S_addr;
else
    /* 如果这个接口没有地址, 那么我们假设这个接口在 C 类网络中 */
    netmask=0xffffffff;

    //编译过滤器规则
    if (pcap_compile(adhandle, &fcode, "ip and tcp", 1, netmask) < 0)
    {
        fprintf(stderr, "Unable to compile the packet filter. Check the syntax.\n");
        /* 释放设备列表 */
        pcap_freealldevs(alldevs);
        return -1;
    }

    //设置过滤器
    if (pcap_setfilter(adhandle, &fcode) < 0)
    {
        fprintf(stderr, "Error setting the filter.\n");
```

```

/* 释放设备列表 */
pcap_freealldevs(alldevs);
return -1;
}

```

2.2.5 发送数据包

Winpcap 提供两种方式发送数据包，一种是单个数据包单独发送，一次发送一个数据包；另一种是连续发送，将待发送的数据包放在发送队列中，按照队列顺序挨个发送。

● 使用 pcap_sendpacket () 发送单个数据包

打开适配器后，调用 pcap_sendpacket()函数来发送一个手写的数据包。pcap_sendpacket()用一个包含要发送的数据的缓冲区、该缓冲区的长度和发送它的适配器作为参数。注意该缓冲区是不经任何处理向外发出的，这意味着，如果想发些有用的东西的话，应用程序必须产生正确的协议头。

int	pcap_sendpacket (pcap_t * p,	//已经打开的适配器
	u_char * str,	//要发送数据包的内容
	int optimize)	//发送数据包的长度

例程：

```

u_char packet[100];
if((fp = pcap_open_live(argv[1], 100, 1, 1000, error) ) == NULL)
{
    fprintf(stderr, "\nError opening adapter: %s\n", error);
    return;
}

packet[0...5]=1;          /* Supposing to be on ethernet, set mac destination to
                           1:1:1:1:1:1 */

packet[6...11]=2;          /* set mac source to 2:2:2:2:2:2 */
for(i=12;i<100;i++)        /* Fill the rest of the packet */
{
    packet[i]=i%256;
}

pcap_sendpacket ( fp, packet, 100); /* Send down the packet */

```

一个完整的打开适配器，发送数据包例子如下：

```

#include <stdlib.h>
#include <stdio.h>
#include <pcap.h>

```

```

void main(int argc, char **argv)
{
    pcap_t *fp;
    char errbuf[PCAP_ERRBUF_SIZE];
    u_char packet[100];
    int i;

    /* 检查命令行参数的合法性 */
    if (argc != 2)
    {
        printf("usage: %s interface (e.g. 'rpcap://eth0')", argv[0]);
        return;
    }

    /* 打开输出设备 */
    if ( (fp= pcap_open(argv[1],                // 设备名
                        100,                    // 要捕获的部分 (只捕获前 100 个字节)
                        PCAP_OPENFLAG_PROMISCUOUS, // 混杂模式
                        1000,                    // 读超时时间
                        NULL,                    // 远程机器验证
                        errbuf                    // 错误缓冲
                        ) ) == NULL)
    {
        fprintf(stderr, "Unable to open the adapter. %s is not supported by WinPcap",
            argv[1]);
        return;
    }

    /* 假设在以太网上, 设置 MAC 的目的地址为 1:1:1:1:1:1 */
    packet[0]=1;
    packet[1]=1;
    packet[2]=1;
    packet[3]=1;
    packet[4]=1;
    packet[5]=1;

    /* 设置 MAC 源地址为 2:2:2:2:2:2 */
    packet[6]=2;
    packet[7]=2;
    packet[8]=2;
    packet[9]=2;
    packet[10]=2;
    packet[11]=2;

```

```

/* 填充剩下的内容 */
for(i=12;i<100;i++)
{
    packet[i]=i%256;
}

/* 发送数据包 */
if (pcap_sendpacket(fp, packet, 100 /* size */) != 0)
{
    fprintf(stderr,"nError sending the packet: n", pcap_geterr(fp));
    return;
}

return;
}

```

- 使用发送队列

`pcap_sendpacket()` 提供了一种简单而直接的方法来发送单个数据包，而 `send_queue()` 则提供了一种更高级，结构更优的方法来发送一组数据包。发送队列是一个容器，它能容纳不同数量的数据包，这些数据包将被发送到网络上。队列的大小表示存储的数据包最大数量。打开适配器后，调用 `send_queue_transmit()` 函数来发送一个队列中的数据包。

发送队列中的关键结构体 `pcap_send_queue()` 定义如下：

```

typedef struct pcap_send_queue( u_int maxlen; //队列最大长度（字节数）
                                u_int len;     //队列当前字节数
                                char *buffer;   //储存待发数据包的缓冲区
                                ) pcap_send_queue;

```

发送队列分 4 个步骤：

- ◆ 发送队列创建
- ◆ 发送队列添加数据包
- ◆ 发送队列
- ◆ 销毁发送队列，释放内存空间

- 发送队列创建函数 `pcap_sendqueue_alloc()`

```

pcap_send_queue *pcap_sendqueue_alloc( u_int memsize)
//参数 memsize 是队列的大小（字节为单位）

```

函数执行成功则返回所分配队列的内存指针，否则返回 NULL。在设置内存空间大小的 memsize 参数时，应注意它包括每个数据包头信息结构体（struct pcap_pkthdr）所占用空间。例如：

```
squeue=pcap_sendqueue_alloc(unsigned int)(( packetlen + sizeof ( struct pcap_pkthdr)) *npacks));
```

上述例子中，packetlen 为数据包的长度，sizeof (struct pcap_pkthdr) 是每个数据包头信息结构体长度，npacks 是数据包个数。

- 添加数据包函数 pcap_sendqueue_queue()

发送队列一旦创建，pcap_sendqueue_queue() 就可以将数据包添加到发送队列中。该函数的原型如下：

```
int pcap_sendqueue_queue(pcap_send_queue *queue, const struct pcap_pkthdr *pkt_header, const u_char *pkt_data )
```

该函数执行成功返回 0，否则返回-1。函数的三个参数分别是：queue 指向 pcap_sendqueue_alloc 函数分配的发送队列；pkt_header 是 winpcap 为每个待发数据包所附加的数据头信息，说明数据包的长度与发送时间戳；pkt_data 为待发数据包。

- 发送队列函数 pcap_sendqueue_transmit()

发送队列创建和添加完毕后，就可使用 pcap_sendqueue_transmit()将队列发送出去。该函数原型定义如下：

```
u_int pcap_sendqueue_transmit(pcap_t *p, pcap_send_queue *queue, int sync )
```

该函数执行成功返回值为实际发送的字节数，如果该值小于希望发送的大小，则表示发送过程中出现错误。函数的 3 个参数的含义如下：p 指向发送数据包的适配器；queue 指向待发送的数据包的队列；sync 为 0 表示尽快发送，不为 0 则根据时间戳发送。

- 释放队列函数 pcap_sendqueue_destroy()

发送队列结束后，不再需要发送队列时，应使用 pcap_sendqueue_destroy() 释放与发送队列有关的所有内存资源，其函数原型如下：

```
void pcap_sendqueue_destroy( pcap_send_queue *queue);
```

使用发送队列发送数据包的完整例子如下所示：

```

/* 分配发送队列 */
squeue = pcap_sendqueue_alloc(caplen);
/* 从文件中将数据包填充到发送队列 */
while ((res = pcap_next_ex( indesc, &pkthdr, &pktdata)) == 1)
{ if (pcap_sendqueue_queue(squeue, pkthdr, pktdata) == -1)
    { printf("Warning: packet buffer too small, not all the packets will be sent.n");
      break;
    }
    npacks++;
}
if (res == -1)
{ printf("Corrupted input file.n");
  pcap_sendqueue_destroy(squeue);
  return;
}

/* 发送队列 */

cpu_time = (float)clock ();
if ((res = pcap_sendqueue_transmit(outdesc, squeue, sync)) < squeue->len)
{ printf("An error occurred sending the packets: %s. Only %d bytes were sentn",
  pcap_geterr(outdesc), res);
}
cpu_time = (clock() - cpu_time)/CLK_TCK;
printf ("nnElapsed time: %5.3fn", cpu_time);
printf ("nTotal packets generated = %d", npacks);
printf ("nAverage packets per second = %d", (int)((double)npacks/cpu_time));
printf ("n");

/* 释放发送队列 */

pcap_sendqueue_destroy(squeue);

```

2.3 Winpcap 分析实例

使用 winpcap 进行网络低层开发的基本步骤和常用函数前面已经做了详细的讲解，本章通过两个网络捕获的实例来练习怎样具体开发网络程序。

2.3.1 捕获解析 UDP 数据包实例

第一个程序的主要目标是解析所捕获的 UDP 数据包的协议首部。UDP 协议首部非常简单，只有 8 个字节，下层网络层使用 IP 协议，作为入门实例，很容易学习。

实例代码如下：

```
#include "pcap.h"

/* 4 字节的 IP 地址 */
typedef struct ip_address{
    u_char byte1;
    u_char byte2;
    u_char byte3;
    u_char byte4;
}ip_address;

/* IPv4 首部 */
typedef struct ip_header{
    u_char ver_ihl;           // 版本 (4 bits) + 首部长度 (4 bits)
    u_char tos;               // 服务类型(Type of service)
    u_short tlen;              // 总长(Total length)
    u_short identification;   // 标识(Identification)
    u_short flags_fo;          // 标志位(Flags) (3 bits) + 段偏移量
                                // (Fragment offset) (13 bits)
    u_char ttl;               // 存活时间(Time to live)
    u_char proto;              // 协议(Protocol)
    u_short crc;               // 首部校验和(Header checksum)
    ip_address saddr;          // 源地址(Source address)
    ip_address daddr;          // 目的地址(Destination address)
    u_int op_pad;              // 选项与填充(Option + Padding)
}ip_header;

/* UDP 首部*/
typedef struct udp_header{
    u_short sport;             // 源端口(Source port)
    u_short dport;             // 目的端口(Destination port)
```

```

        u_short len;                // UDP 数据包长度(Datagram length)
        u_short crc;                // 校验和(Checksum)
    }udp_header;

                                /* 回调函数原型 */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char
*pkt_data);

main()
{
    pcap_if_t *alldevs;
    pcap_if_t *d;
    int inum;
    int i=0;
    pcap_t *adhandle;
    char errbuf[PCAP_ERRBUF_SIZE];
    u_int netmask;
    char packet_filter[] = "ip and udp";
    struct bpf_program fcode;

                                /* 获得设备列表 */
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
    {
        fprintf(stderr,"Error in pcap_findalldevs: %s", errbuf);
        exit(1);
    }

                                /* 打印列表 */

    for(d=alldevs; d; d=d->next)
    {
        printf("%d. %s", ++i, d->name);
        if (d->description)
            printf(" (%s)", d->description);
        else
            printf(" (No description available)");
    }

    if(i==0)
    {
        printf("\nNo interfaces found! Make sure WinPcap is installed.\n");
        return -1;
    }

    printf("Enter the interface number (1-%d):",i);

```

```

scanf("%d", &inum);

if(inum < 1 || inum > i)
{
printf("Interface number out of range.n");
/* 释放设备列表 */

pcap_freealldevs(alldevs);
return -1;
}

/* 跳转到已选设备 */
for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);

/* 打开适配器 */
if ( (adhandle= pcap_open(d->name,
                          65536,
                          PCAP_OPENFLAG_PROMISCUOUS,
                          1000,
                          NULL,
                          errbuf)
    ) == NULL)
{
    // 设备名
    //捕获数据包的长度限制
    // 混杂模式
    // 读取超时时间
    // 远程机器验证
    // 错误缓冲池
    fprintf(stderr,"Unable to open the adapter. %s is not supported by WinPcapn");
    /* 释放设备列表 */

    pcap_freealldevs(alldevs);
    return -1;
}

/* 检查数据链路层，只考虑以太网 */
if(pcap_datalink(adhandle) != DLT_EN10MB)
{
    fprintf(stderr,"This program works only on Ethernet networks.n");
    /* 释放设备列表 */
    pcap_freealldevs(alldevs);
    return -1;
}

if(d->addresses != NULL)
/* 获得接口第一个地址的掩码 */
netmask=((struct sockaddr_in *)(d->addresses->netmask))->sin_addr.S_un.S_addr;
else
/* 如果接口没有地址，那么我们假设一个 C 类的掩码 */
netmask=0xfffff;

```

```

//编译过滤器
if (pcap_compile(adhandle, &fcode, packet_filter, 1, netmask) <0 )
{
fprintf(stderr,"nUnable to compile the packet filter. Check the syntax.n");
/* 释放设备列表 */

pcap_freealldevs(alldevs);
return -1;
}

//设置过滤器

if (pcap_setfilter(adhandle, &fcode)<0)
{
fprintf(stderr,"nError setting the filter.n");
/* 释放设备列表 */

pcap_freealldevs(alldevs);
return -1;
}

printf("nlistening on %s...n", d->description);

/* 释放设备列表 */

pcap_freealldevs(alldevs);

/* 开始捕捉 */

pcap_loop(adhandle, 0, packet_handler, NULL);

return 0;
}

/* 回调函数，当收到每一个数据包时会被 libpcap 所调用 */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char
*pkt_data)
{
struct tm *ltime;
char timestr[16];
ip_header *ih;
udp_header *uh;
u_int ip_len;
u_short sport,dport;
time_t local_tv_sec;

/* 将时间戳转换成可识别的格式 */
local_tv_sec = header->ts.tv_sec;
ltime=localtime(&local_tv_sec);

```

```

strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);

/* 打印数据包的时间戳和长度 */
printf("%s.%.6d len:%d ", timestr, header->ts.tv_usec, header->len);

/* 获得 IP 数据包头部的位置 */
ih = (ip_header *) (pkt_data +
14); //以太网头部长度

/* 获得 UDP 首部的位置 */
ip_len = (ih->ver_ihl & 0xf) * 4;
uh = (udp_header *) ((u_char*)ih + ip_len);

/* 将网络字节序列转换成主机字节序列 */
sport = ntohs( uh->sport );
dport = ntohs( uh->dport );

/* 打印 IP 地址和 UDP 端口 */
printf("%d.%d.%d.%d.%d -> %d.%d.%d.%d.%dn",
ih->saddr.byte1,
ih->saddr.byte2,
ih->saddr.byte3,
ih->saddr.byte4,
sport,
ih->daddr.byte1,
ih->daddr.byte2,
ih->daddr.byte3,
ih->daddr.byte4,
dport);
}

```

首先，我们将过滤器设置成"ip and udp"。在这种方式下，我们确信 packet_handler() 只会收到基于 IPv4 的 UDP 数据包；这将简化解析过程，提高程序的效率。

packet_handler(), 尽管只受限于单个协议的解析（比如基于 IPv4 的 UDP），不过它展示了捕捉器(sniffers)是多么的复杂，就像 TcpDump 或 WinDump 对网络数据流进行解码那样。因为我们对 MAC 首部不感兴趣，所以我们跳过它。为了简洁，我们在开始捕捉前，使用了 pcap_datalink() 对 MAC 层进行了检测，以确保我们是在处理一个以太网络。这样，我们就能确保 MAC 首部是 14 位的。

IP 数据包的首部就位于 MAC 首部的后面。我们将从 IP 数据包的首部解析到源 IP 地址和目的 IP 地址。

处理 UDP 的首部有一些复杂，因为 IP 数据包的首部的长度并不是固定的。然而，我们可以通过 IP 数据包的 length 域来得到它的长度。一旦我们知道了 UDP 首部的位置，我们就能解析到源端口和目的端口。

被解析出来的值被打印在屏幕上，形式如下所示：

```
1. DevicePacket_{A7FD048A-5D4B-478E-B3C1-34401AC3B72F} (Xircom t
10/100 Adapter)
Enter the interface number (1-2):1
```

```
listening on Xircom CardBus Ethernet 10/100 Adapter...
16:13:15.312784 len:87 130.192.31.67.2682 -> 130.192.3.21.53
16:13:15.314796 len:137 130.192.3.21.53 -> 130.192.31.67.2682
16:13:15.322101 len:78 130.192.31.67.2683 -> 130.192.3.21.53
```

2.3.2 打印通过适配器的数据包实例

第二个实例程序将每一个通过适配器的数据包打印出来。仅仅打印经过适配器的数据包的统计信息。

```
-----
#include "pcap.h"

/* packet handler 函数原型 */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char
*pkt_data);

main()
{
    pcap_if_t *alldevs;
    pcap_if_t *d;
    int inum;
    int i=0;
    pcap_t *adhandle;
    char errbuf[PCAP_ERRBUF_SIZE];

                                                                    /* 获取本机设备列表 */
```

```

if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
{
fprintf(stderr, "Error in pcap_findalldevs: %sn", errbuf);
exit(1);
}

/* 打印列表 */

for(d=alldevs; d; d=d->next)
{
printf("%d. %s", ++i, d->name);
if (d->description)
printf(" (%s)n", d->description);
else
printf(" (No description available)n");
}

if(i==0)
{
printf("nNo interfaces found! Make sure WinPcap is installed.n");
return -1;
}

printf("Enter the interface number (1-%d):", i);
scanf("%d", &inum);

if(inum < 1 || inum > i)
{
printf("nInterface number out of range.n");

/* 释放设备列表 */

pcap_freealldevs(alldevs);
return -1;
}

/* 跳转到选中的适配器 */

for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);

/* 打开设备 */
// 设备名
// 数据包的最大长度
// 混杂模式
// 读取超时时间
// 远程机器验证
// 错误缓冲池

if ( (adhandle= pcap_open( d->name,
                           65536,
                           PCAP_OPENFLAG_PROMISCUOUS,
                           1000,
                           NULL,
                           errbuf)
    ) == NULL)
{

```

```

fprintf(stderr,"Unable to open the adapter. %s is not supported by WinPcapn",
d->name);

/* 释放设备列表 */

pcap_freealldevs(alldevs);
return -1;
}
printf("nlistening on %s...n", d->description);

/* 释放设备列表 */

pcap_freealldevs(alldevs);

/* 开始捕获 */

pcap_loop(adhandle, 0, packet_handler, NULL);

return 0;
}

/* 每次捕获到数据包时，libpcap 都会自动调用这个回调函数 */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char
*pkt_data)
{
    struct tm *ltime;
    char timestr[16];
    time_t local_tv_sec;

    /* 将时间戳转换成可识别的格式 */

    local_tv_sec = header->ts.tv_sec;
    ltime=localtime(&local_tv_sec);
    strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);

    printf("%s,%.6d len:%dn", timestr, header->ts.tv_usec, header->len);

}

```

上面的程序将每一个数据包的时间戳和长度从 `pcap_pkthdr` 的首部解析出来，并打印在屏幕上。

3 WinSocket 网络编程技术

3.1 WinSocket 概述

Windows Socket 是从 UNIX Socket 继承发展而来，最新的版本是 2.2。进行 Windows 网络编程，你需要在你的程序中包含 `WINSOCK2.H` 或 `MSWSOCK.H`，同时你需要添加引入库 `WS2_32.LIB` 或 `WSOCK32.LIB`。

Socket 编程有阻塞和非阻塞两种，在操作系统 I/O 实现时又有几种模型，包括 `Select`，`WSAAsyncSelect`，`WSAEventSelect`，IO 重叠模型，完成端口等。学习基本的网络编程概念，可以选择从阻塞模式开始，而要开发真正实用的程序，就要进行非阻塞模式的编程（大型服务器通常不采用阻塞模式进行网络通信）。在选择 I/O 模型时，可以从 `WSAAsyncSelect` 模型开始，因为它比较简单，而且有一定的实用性。

3.2 Winsocket 开发流程

Socket 是网络应用层和传输层的接口，传输层提供两种不同的服务：一个是面向连接，可靠的 TCP 协议；另一个是无连接的，不可靠的 UDP 协议。针对这两种服务，Socket 的开发流程略有不同。

面向无连接的 Socket 开发流程如下图所示：

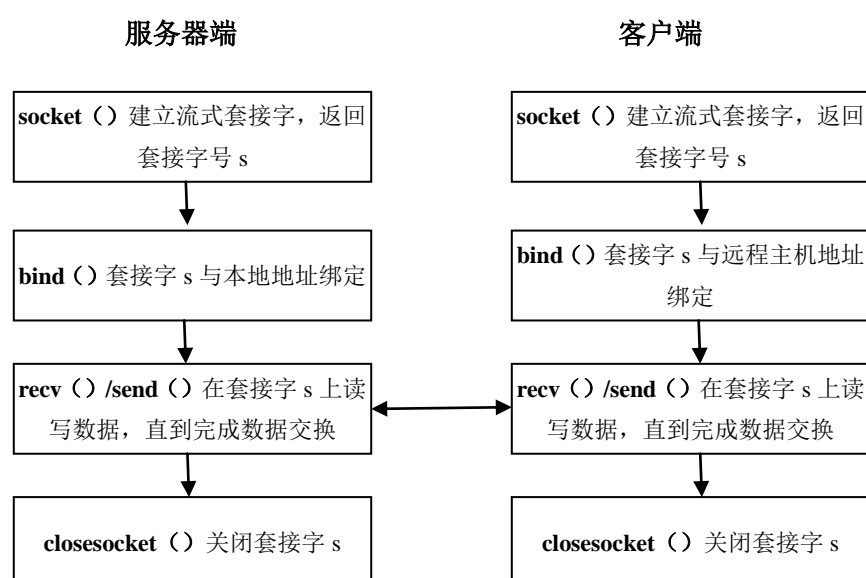


图 4-1 面向无连接的 Socket 流程

面向连接的 Socket 开发流程如下图所示：

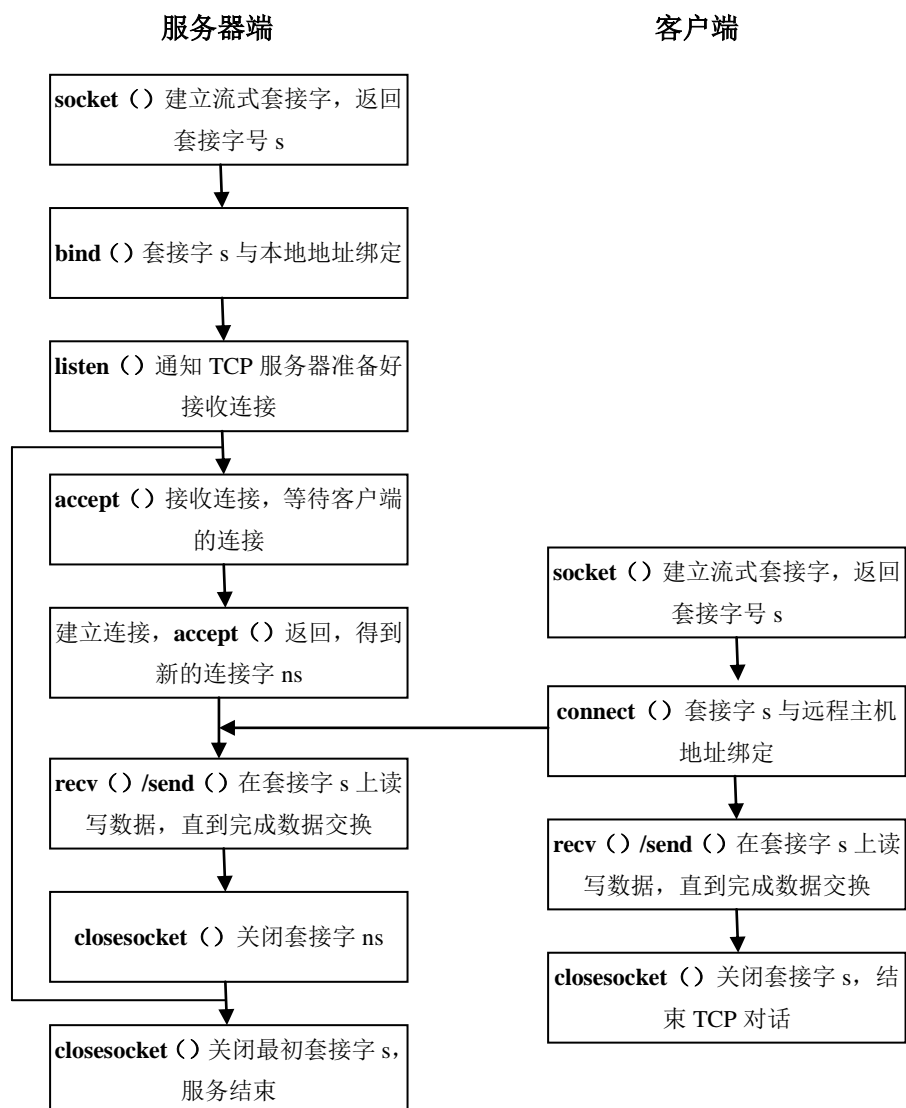


图 4-2 面向连接的 socket 流程

3.3 Winsocket 基本函数

● 创建套接字——`socket()`

功能：创建一个新的套接字，格式如下：

SOCKET PASCAL FAR `socket(int af,int type,int proctol);`

参数： `af`: 通信发生的区域

`type`: 要建立的套接字类型

`proctol`: 使用的特定协议

- 指定本地地址——bind()

功能：将套接字地址与所创建的套接字号联系起来，格式如下：

```
int PASCAL FAR bind(SOCKET s,const struct sockaddr FAR * name,int namelen);
```

参数：s：是由 socket()调用返回的并且未作连接的套接字描述符；

其它：没有错误，bind()返回 0，否则 SOCKET_ERROR

其中的地址结构说明：

```
struct sockaddr_in
{
    short sin_family;           //AF_INET
    u_short sin_port;           //16 位端口号，网络字节顺序
    struct in_addr sin_addr;     //32 位 IP 地址，网络字节顺序
    char sin_zero[8];           //保留
}
```

- 建立套接字连接——connect()和 accept()

功能：共同完成连接工作，格式如下：

```
int PASCAL FAR connect(SOCKET s,const struct sockaddr FAR * name,int
namelen);
```

```
SOCKET PASCAL FAR accept(SOCKET s,struct sockaddr FAR * name,int FAR *
addrlen);
```

参数：同上

- 监听连接——listen()

功能：用于面向连接服务器，表明它愿意接收连接，格式如下：

```
int PASCAL FAR listen(SOCKET s, int backlog);
```

- 数据传输——send()与 recv()

功能：数据的发送与接收，格式如下：

```
int PASCAL FAR send(SOCKET s,const char FAR * buf,int len,int flags);
```

```
int PASCAL FAR recv(SOCKET s,const char FAR * buf,int len,int flags);
```

参数: buf:指向存有传输数据的缓冲区的指针。

- 多路复用——select()

功能: 用来检测一个或多个套接字状态, 格式如下:

```
int PASCAL FAR select(int nfd, 错误!未找到引用源。错误!未找到引用源。fd_set FAR * readfds,fd_set FAR * writefds, fd_set FAR * exceptfds,const struct timeval FAR * timeout);
```

参数: readfds:指向要做读检测的指针;

writefds:指向要做写检测的指针

exceptfds:指向要检测是否出错的指针

timeout:最大等待时间

- 同步 I/O 多路复用——select()*

功能: 用来同步套接字, 格式如下:

```
select ( int nfd, fd_set readfds, fd_set writefds, fd_set exceptfds, const struct timeval timeout )
```

参数: nfd: 设成 0;

FD_SET readfds: 被检查是否可读的 Sockets (要接收数据的套接字);

FD_SET ritefds: 被检查是否可写的 Sockets (要发送数据的套接字);

FD_SET exceptfds: 被检查是否有错误的 Sockets select()

timeout: 设定 select 函数要等待 (block) 多久。

- 关闭套接字——closesocket()

功能: 关闭套接字 s, 格式如下:

```
BOOL PASCAL FAR closesocket(SOCKET s);
```

3.4 Winsockets 编程实例

如上述开发流程所示, 服务器端和客户端开发相应的应用程序。下面是一个 Socket 编程的 C/S 结构的网络应用程序实例。

Socket 服务器端流程如下: 加载套接字->创建监听的套接字->绑定套接字->监听套接字->处理客户端相关请求。

```
#include <Winsock2.h>
```

```

#include <stdio.h>

void main()
{
    //加载套接字

    WORD wVersionRequested;

    WSADATA wsaData;

    int err;

    wVersionRequested=MAKEWORD(1,1);

    err=WSAStartup(wVersionRequested,&wsaData);

    if (err!=0)
    {
        return;
    }

    if (LOBYTE(wsaData.wVersion)!=1|| HIBYTE(wsaData.wVersion)!=1)
    {
        WSACleanup();

        return;
    }

    //创建监听的套接字

    SOCKET sockSrv=socket(AF_INET,SOCK_STREAM,0);

    SOCKADDR_IN addrSrv;

    addrSrv.sin_addr.S_un.S_addr=htonl(INADDR_ANY);    //把 U_LONG 的主
    机字节顺序转换为 TCP/IP 网络字节顺序

    addrSrv.sin_family=AF_INET;

    addrSrv.sin_port=htons(6000);

    //绑定套接字

    bind(sockSrv,(SOCKADDR*)&addrSrv,sizeof(SOCKADDR));

    //将套接字设置为监听模式，准备接受用户请求

```

```

listen(sockSrv,5);

SOCKADDR_IN addrClient;

int len=sizeof(SOCKADDR);

printf("%s\n","welcome,the serve is started...");

while (1)
{
    //等待用户请求到来

    SOCKET sockConn=accept(sockSrv,(SOCKADDR*)&addrClient,&len);

    char sendBuf[100];

    sprintf(sendBuf,"welcome                %s                to
http://unblue2008.javaeye.com",inet_ntoa(addrClient.sin_addr));

                                //发送数据

    send(sockConn,sendBuf,100,0);

    char revBuf[100];

                                //接收数据

    recv(sockConn,revBuf,100,0);

                                //打印接受数据

    printf("%s\n",revBuf);

                                //关闭套接字

    closesocket(sockConn);
}
}

```

Socket 客户端同样需要先加载套接字，然后创建套接字，不过之后不用绑定和监听了，而是直接连接服务器，发送相关请求。Socket 客户端程序如下：

```

#include <Winsock2.h>

#include <stdio.h>

void main()

```

```

{                                                                    //加载套接字

    WORD wVersionRequested;

    WSADATA wsaData;

    int err;

    wVersionRequested=MAKEWORD(1,1);

    err=WSAStartup(wVersionRequested,&wsaData);

    if (err!=0)

    {

        return;

    }

    if (LOBYTE(wsaData.wVersion)!=1|| HIBYTE(wsaData.wVersion)!=1)

    {

        WSACleanup();

        return;

    }

                                                                    //创建套接字

    SOCKET sockClient=socket(AF_INET,SOCK_STREAM,0);

    SOCKADDR_IN addrSrv;

    addrSrv.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");    //把 U_LONG 的主
    机字节顺序转换为 TCP/IP 网络字节顺序

    addrSrv.sin_family=AF_INET;

    addrSrv.sin_port=htons(6000);

                                                                    //向服务器发送请
    求

    connect(sockClient,(SOCKADDR*)&addrSrv,sizeof(SOCKADDR));

                                                                    //接受数据

    char recBuf[100];

```

```
recv(sockClient,recBuf,100,0);

printf("%s\n",recBuf);

//发送数据
send(sockClient,"this is 客户机",strlen("this is 客户机")+1,0);

//关闭套接字
closesocket(sockClient);

WSACleanup();
}
```