实验 10 Netfilter 框架编程

目录

实验目的

了解 NetFilter 框架，掌握 Linux 内核模块编程的方法，熟悉 Netfilter 框架进行数据拦截的方法，能够利用 Netfilter 框架实现网络数据包的控制。

实验要求

1. 掌握 Linux 内核模块的编程方法
2. 掌握 Netfilter 框架钩子函数设计与实现方法
3. 利用 Netfilter 框架和钩子函数实现以下功能
   - 禁止 ping 发送
   - 禁止某个 IP 地址数据包的接受
   - 禁止某个端口的数据响应

实验环境

实验环境采用 **vscode remote + wsl**，操作系统的版本为 **Ubuntu 20.04.2 LTS**，如下图所示

```
❯ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.2 LTS
Release:        20.04
Codename:       focal
```

操作系统版本

内核版本如下图所示

```
❯ uname -a
Linux qiufeng 5.10.16.3-microsoft-standard-WSL2
```

内核版本

gcc 版本为 9.3.0

```
❯ gcc --version
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

gcc 版本

需要注意的是，由于在 **wsl2** 中不存在 linux 源码，因此为了编译内核模块，

我们必须手动下载 linux 源码，并进行编译

```
git clone https://github.com/microsoft/WSL2-Linux-Kernel.git --depth 1
sudo apt install build-essential flex bison libssl-dev libelf-dev
cd WSL2-Linux-Kernel
cp Microsoft/config-wsl .config
sudo make -j $(expr $(nproc) - 1))
```

下载并编译源码

接着需要在 User 根目录下创建**.wslconfig** 文件，写入如下内容

```
[wsl2]
kernel=C:\\Users\\<USERNAME>\\bzimage
```

wslconfig

并 将 **./WSL2-Linux-Kernel/arch/x86/boot/bzimage** 移 动 到 **C:\Users\<USERNAME>**文件夹中

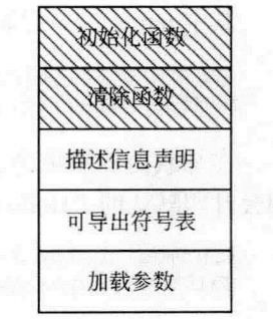
最后需要重启 **wsl**

实验步骤

1. Linux 内核模块的编程方法

1.1 内核模块的基本结构

Linux 内核模块的基本结构如下图所示



内核模块基本结构

其中各个模块的作用为

- **初始化函数**：进行模块的初始化工作，主要是资源的申请
- **清除函数**：进行模块的清除工作，主要是状态重置和资源释放
- **描述信息声明**：内核的一些描述性信息，包括模块作者、模块用途、模块版本号和开源许可证等
- **可导出符号表**：将允许导出的符号加到公共内核符号表中，或者使用公共内核符号表来解析加载模块中未定义的符号
- **加载参数**：定义在内核模块加载时传递的参数

1.2 初始化和清除函数

在内核模块的编写过程中，**初始化函数**和**清除函数**是必须具备的结构。

其中初始化函数的主要流程为

- 使用 *insmod* 加载内核模块
- 调用 *init_module* 函数
- 进入到系统调用 *sys_init_module* 函数。该函数负责将模块程序复制到内核中，进行必要的检查、分配资源等
- 调用内核模块中的 *init* 函数

初始化函数的代码形式如下

```c
static int __init initialize(void) {
    /*
        内核初始化代码
    */
    return 0;
}

module_init(initialize);
```

内核初始化函数

清除函数的主要流程为

- 使用 *rmmod* 卸载内核模块
- 调用 *delete_module* 函数
- 进入到系统调用 *sys_delete_module* 函数。该函数负责卸载前检查、内核清理函数的调用、模块清理等工作

清除函数的代码形式如下

```c
static void __exit exit(void) {
    /*
        内核清理代码
    */
}

module_exit(exit);
```

内核清除函数

1.3 Makefile 文件编写

最后，为了编译内核模块，我们需要编写如下所示的 Makefile 代码

```
obj-m += filter.o

all:
    make -C $(shell pwd)/WSL2-Linux-Kernel M=$(shell pwd) modules

clean:
    make -C $(shell pwd)/WSL2-Linux-Kernel M=$(shell pwd) clean
```

目标文件

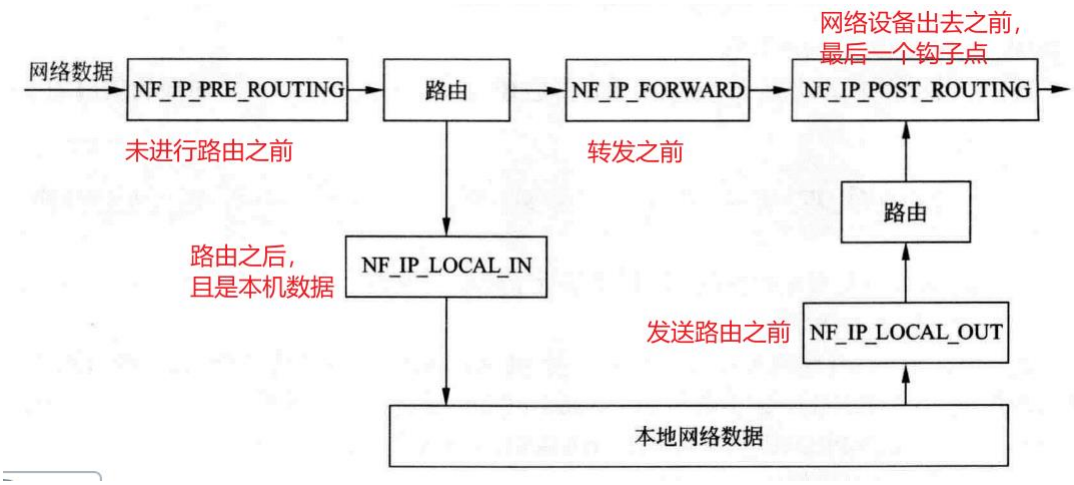
内核代码路径

Makefile

2. Netfilter 框架钩子函数设计与特定功能实现

2.1 钩子点介绍

在 Linux 内核中，netfilter 一共有 5 个钩子点，分别是 ***PREROUTING***、***POSTROUTING***、***INPUT***、***FORWARD***、***OUTPUT***，其所在位置和含义如下图所示



钩子点

当网络数据包到来时，内核通过 *ip_rcv* 函数接受，在函数的末尾会调用 *NF_HOOK* 将控制权交给 *NF_IP_PRE_ROUTING* 处的钩子。

若网络数据包是发送到本地的，则会调用 *ip_local_deliver* 函数，该函数在末尾调用 *NF_HOOK* 并将控制权交给 *NF_IP_LOCAL_IN* 处的钩子

若网络数据是转发，则会调用 *ip_forward* 函数，该函数在末尾将控制权交

给 *NF_IP_FORWARD* 处的钩子

通过调用 *ip_send* 函数发送数据包，在该函数发送数据之前会将控制权交给 *NF_IP_POST_ROUTING* 处的钩子

若本机要发送网络数据，则会调用 *NF_IP_LOCAL_OUT* 处的钩子

2.2 钩子处理规则介绍

钩子函数的返回值可以为 *NF_ACCEPT*、*NF_DROP*、*NF_STOLEN*、*NF_QUERE*、*NF_REPEAT*，分别对应 5 种处理规则，具体的含义如下所示

- *NF_ACCEPT*：继续传递，保持和原来的传输一致
- *NF_DROP*：丢弃包
- *NF_STOLEN*：接管包，不再继续传递
- *NF_QUERE*：队列化包
- *NF_REPEAT*：再次调用这一个钩子

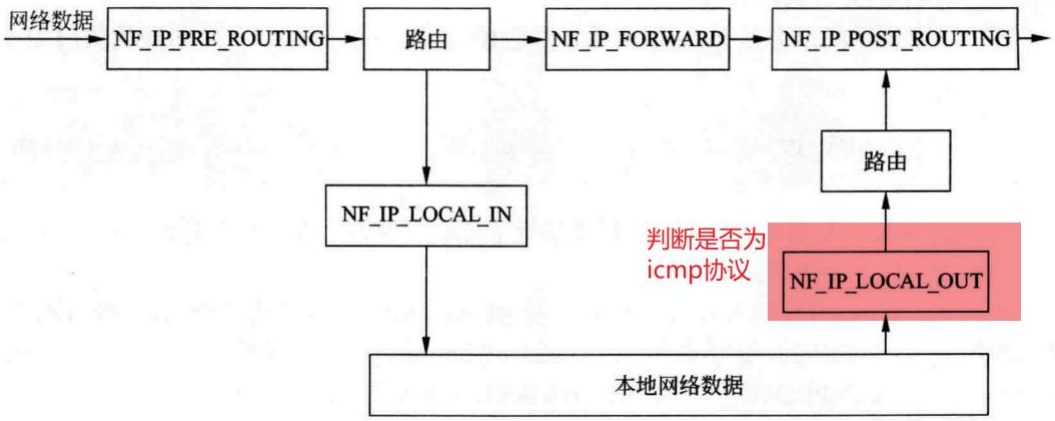
2.3 特定功能的设计和实现

一共包含 4 个模块，分别是 **Ping 模块**、**IP 模块**、**端口模块**、**动态配置模块**

2.3.1 Ping 模块

Ping 模块的主要功能为**禁止 ping 发送**。

分析要求可知，我们只需要在 *NF_IP_LOCAL_OUT* 处挂载钩子函数



钩子点

钩子函数的处理流程可用下图表示

- 获取 IP 头部
- 判断是否为 icmp 协议
- 若是，则丢弃该数据包

处理流程

对应代码如下所示

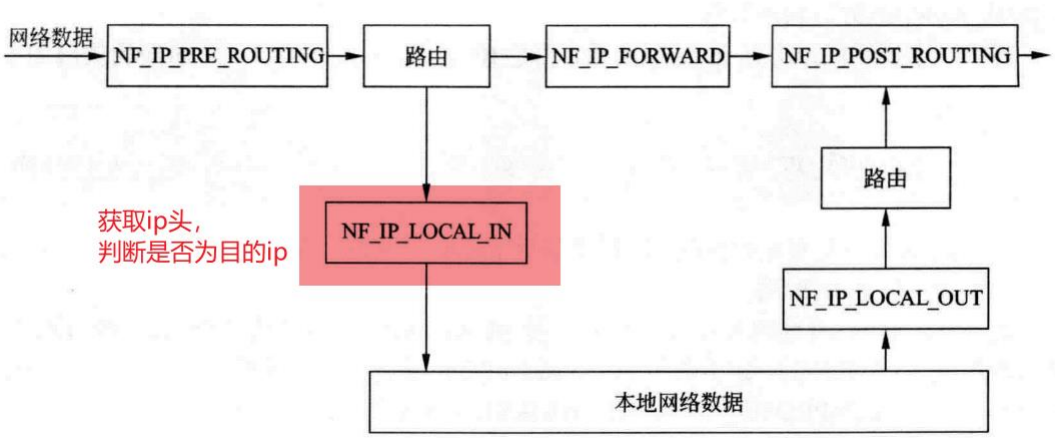
```c
//* 获取 IP 头部
iph = ip_hdr(skb);
unsigned int dest_ip = iph->daddr;
//! 判断是否为 ICMP
if (iph->protocol == IPPROTO_ICMP)
{
    //* 判断当前是否为禁止状态，若是，则丢弃 ICMP 报文
    if (IS_BANDPING(b_status))
    {
        // 注意是大端对齐 字节序
        printk(KERN_ALERT "DROP ICMP packet from %d.%d.%d.%d\n",
               (dest_ip & 0x000000ff) >> 0,
               (dest_ip & 0x0000ff00) >> 8,
               (dest_ip & 0x00ff0000) >> 16,
               (dest_ip & 0xff000000) >> 24);

        return (NF_DROP); /*丢弃该报文*/
    }
}

return (NF_ACCEPT);
```

禁止 Ping 发送

2.3.2 IP 模块

IP 模块的主要功能为**禁止某个 IP 地址数据包的接受**。

分析要求可知，我们只需要在 ***NF_IP_LOCAL_IN*** 处挂载钩子函数

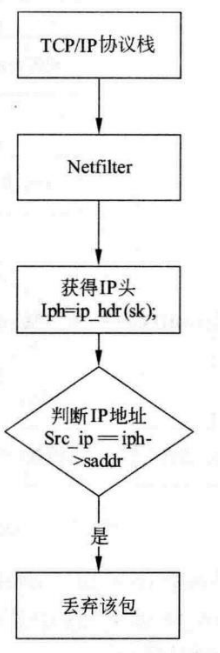


钩子点

钩子函数的处理流程可以用下图来表示

- 获取 IP 头部

- 判断源 IP 是否为要禁止的 IP 地址，若是，则丢弃该数据包



处理流程

对应代码如下所示

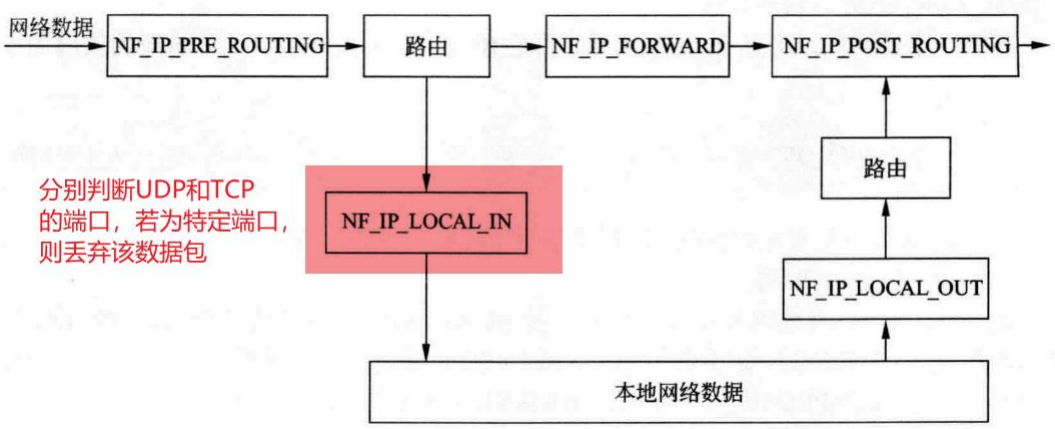
```
//* 获取 IP 头
iph = ip_hdr(skb);
//* 获取源 IP 地址
unsigned int src_ip = iph->saddr;
//! 判断是否禁止 IP
if (IS_BANDIP(b_status))
{
    //! 若源 IP 地址与被禁止的 IP 相等，则丢弃数据包
    if (b_status.band_ip == iph->saddr)
    {
        printk(KERN_ALERT "DROP IP packet from %d.%d.%d.%d\n",
               (src_ip & 0x000000ff) >> 0,
               (src_ip & 0x0000ff00) >> 8,
               (src_ip & 0x00ff0000) >> 16,
               (src_ip & 0xff000000) >> 24);
        return (NF_DROP);
    }
}
```

禁止特定 IP 接受

2.3.3 端口模块

端口模块的主要功能为**禁止某个端口的数据响应**。

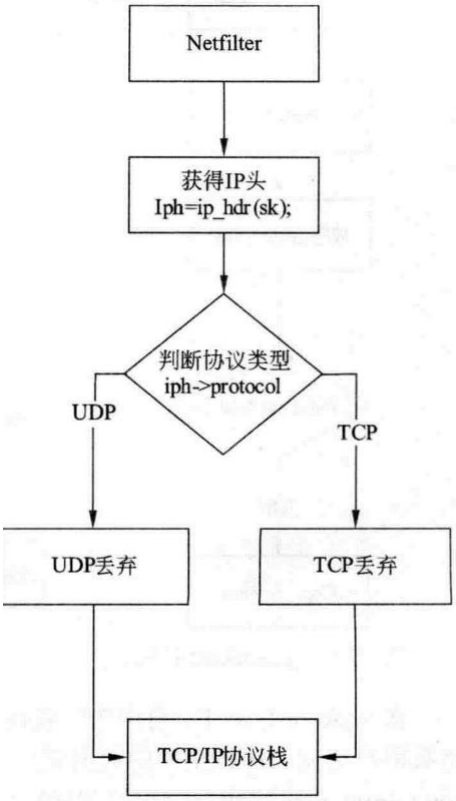
分析要求可知，我们只需要在 ***NF_IP_LOCAL_IN*** 处挂载钩子函数



钩子点

钩子函数的处理流程可以用下图来表示

- 获取 IP 头部
- 判断传输层协议类型
- 如果为 TCP，且目的端口为禁止端口，则丢弃该数据包
- 如果为 UDP，且目的端口为禁止端口，则丢弃该数据包



处理流程

对应代码如下图所示

```c
//! 判断传输层协议类型
switch (iph->protocol)
{
    //* 如果为 TCP 协议
    case IPPROTO_TCP:
        //* 如果开启了禁止 tcp 端口响应
        if (IS_BANDPORT_TCP(b_status))
        {
            //* 获取 tcp 头部
            tcph = tcp_hdr(skb);
            //! 如果目的端口库为要禁止的端口，则丢弃数据包
            if (tcph->dest == b_status.band_port.port)
            {

                printk(KERN_ALERT "DROP TCP port %d IP packet from %d.%d.%d.%d\n",
                    port_local,
                    (src_ip & 0x000000ff) >> 0,
                    (src_ip & 0x0000ff00) >> 8,
                    (src_ip & 0x00ff0000) >> 16,
                    (src_ip & 0xff000000) >> 24);
                return (NF_DROP);
            }
        }
        break;
    //* 如果为 UDP 协议
    case IPPROTO_UDP:
        //* 如果开启了禁止 udp 端口响应
        if (IS_BANDPORT_UDP(b_status))
        {
            //* 获取 udp 头部
            udph = udp_hdr(skb);
            //! 如果目的端口库为要禁止的端口，则丢弃数据包
            if (udph->dest == b_status.band_port.port) /*UDP端口判定*/
            {
                printk(KERN_ALERT "DROP UDP port %d IP packet from %d.%d.%d.%d\n",
                    port_local,
                    (src_ip & 0x000000ff) >> 0,
                    (src_ip & 0x0000ff00) >> 8,
                    (src_ip & 0x00ff0000) >> 16,
                    (src_ip & 0xff000000) >> 24);
                return (NF_DROP);
            }
        }
        break;
```

禁止某个端口的数据响应

2.3.4 动态配置模块

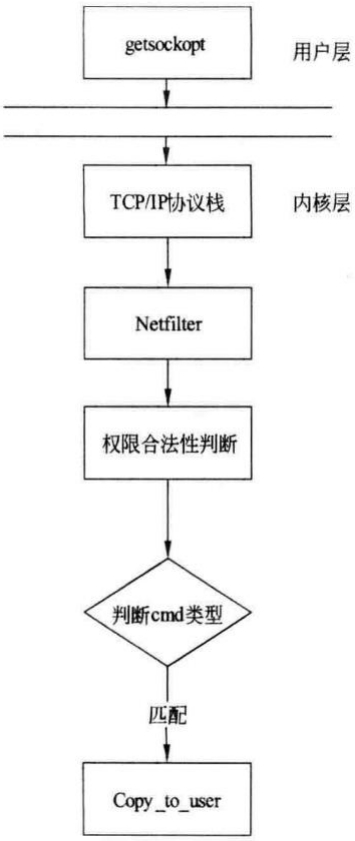
动态配置模块的主要功能为**通过内核交互实现配置的动态改变**。

主要通过 ***getsockopt*** 和 ***setsockopt*** 扩展实现内核交互

其中 ***getsockopt*** 的主要流程如下图所示

- 判断用户权限合法性

- 判断 cmd 类型合法性
- 读取配置到用户程序



getsockopt 主要流程

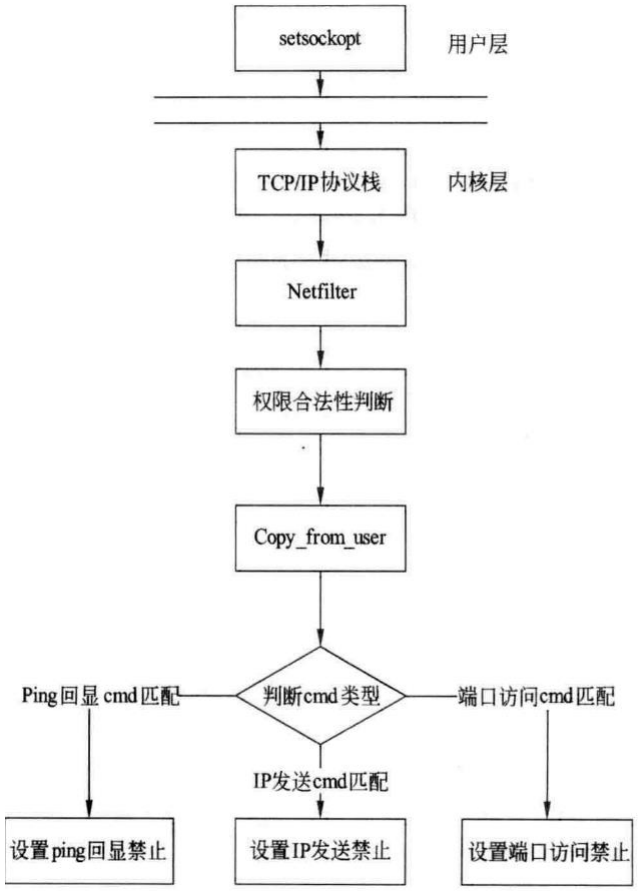
对应代码如下

```c
int ret = 0;
//! 权限检查
if (!capable(CAP_NET_ADMIN))
{
    ret = -EPERM;
    goto ERROR;
}
//! cmd 类型判断
switch (cmd)
{
case SOE_BANDIP:
case SOE_BANDPORT:
case SOE_BANDPING:
    //! 将数据从内核空间复制到用户空间
    ret = copy_to_user(user, &b_status, *len);
    if (ret != 0)
    {
        ret = -EINVAL;
        goto ERROR;
    }
    break;
default:
    ret = -EINVAL;
    break;
}
```

getsockopt 对应代码

setsockopt 的主要流程为

- 用户权限合法性判断
- 将用户输入参数复制到内核空间
- 根据 cmd 类型配置相应的模块



setsockopt 主要流程

对应代码如下

用户权限检查和从用户空间复制数据

```
int ret = 0;
struct band_status status;

//! 权限检查
if (!capable(CAP_NET_ADMIN))
{
    ret = -EPERM;
    goto ERROR;
}
//! 从用户空间复制数据
ret = copy_from_user(&status, user, len);
if (ret != 0) /*复制数据失败*/
{
    ret = -EINVAL;
    goto ERROR;
}
```

权限检查和从用户空间复制数据

配置 IP 模块

```
//! IP模块配置
case SOE_BANDIP:
    //* 禁止 IP
    if (IS_BANDIP(status))
    {
        b_status.band_ip = status.band_ip;
        printk(KERN_ALERT "IP地址%d", b_status.band_ip);
    }
    //* 取消禁止 IP
    else
    {
        b_status.band_ip = 0;
        printk(KERN_ALERT "IP地址为0");
    }
```

配置 IP 模块

配置端口模块

```
//! 端口模块配置
case SOE_BANDPORT:
    //* 禁止 tcp 端口
    if (IS_BANDPORT_TCP(status))
    {
        b_status.band_port.protocol = IPPROTO_TCP;
        b_status.band_port.port = status.band_port.port;
    }
    //* 禁止 udp 端口
    else if (IS_BANDPORT_UDP(status))
    {
        b_status.band_port.protocol = IPPROTO_UDP;
        b_status.band_port.port = status.band_port.port;
    }
    //* 取消禁止端口
    else
    {
        b_status.band_port.protocol = 0;
        b_status.band_port.port = 0;
    }
```

配置端口模块

配置 ping 模块

```c
//! ping 模块配置
case SOE_BANDPING:
    //* 禁止 ping
    if (IS_BANDPING(status))
    {
        b_status.band_ping = 1;
    }
    //* 取消禁止 ping
    else
    {
        b_status.band_ping = 0;
    }
```

配置 ping 模块

实验测试

执行命令 ***sudo make all*** 编译内核模块

```
❯ sudo make all
make -C /home/qiufeng/courses/networking-programming/10/WSL2-Linux-Kernel M=/home/qiufeng/courses/networking-program
ming/10 modules -w
make[1]: Entering directory '/home/qiufeng/courses/networking-programming/10/WSL2-Linux-Kernel'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/home/qiufeng/courses/networking-programming/10/WSL2-Linux-Kernel'
```

编译内核模块

执行命令 ***sudo insmod filter.ko*** 加载内核模块，并通过 ***dmesg*** 查看内核模块输出

```
❯ sudo insmod filter.ko
❯ dmesg
[ 5259.416348] filter: loading out-of-tree module taints kernel.
[ 5259.418129] netfilter example 2 install successfully
[ 5264.913904] WSL2: Performing memory compaction.
```

加载内核模块

1. 内核交互测试

执行交互程序***./test***，发现能够正常和内核进行交互

```
❯ sudo ./test
打开设备
读取设备输出内核信息


band ping
band TCP port 80
no band ip
设置内核
输入xx.xx.xx.xx以禁止IP:
```

执行测试程序

在交互程序中进行如下配置

```
❯ sudo ./test
打开设备
读取设备输出内核信息


band ping
no band port
123.123.123.123
设置内核
输入xx.xx.xx.xx以禁止IP:123.123.123.123
2071690107
2071690107
123.123.123.123
禁止ping(Y/other):y
禁止TCP端口(Y/other):禁止UDP端口(Y/other):y
输入端口:8080
输出内核信息


band ping
band UDP port 8080
123.123.123.123
```

设置禁止的ip地址

禁止ping

禁止端口数据接受

进行相关的配置

2. ping 模块测试

只开启 ping 模块

```
❯ sudo ./test
打开设备
读取设备输出内核信息

band ping
band UDP port 8080
123.123.123.123
设置内核
输入xx.xx.xx.xx以禁止IP:
禁止ping(Y/other):y
禁止TCP端口(Y/other):禁止UDP端口(Y/other):
输出内核信息

band ping
no band port
no band ip
```

当前配置

新配置，只禁止ping

只开启 ping 模块

尝试 ping 百度服务器，可以发现被成功禁止了

```
❯ ping baidu.com
PING baidu.com (220.181.38.148) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```

ping 被禁止

内核输出如下

```
[ 5807.305615] DROP ICMP packet from 220.181.38.148
[ 5808.332045] DROP ICMP packet from 220.181.38.148
[ 5809.372068] DROP ICMP packet from 220.181.38.148
[ 5810.412039] DROP ICMP packet from 220.181.38.148
[ 5811.452161] DROP ICMP packet from 220.181.38.148
```

内核输出

3. IP 模块测试

未开启模块之前可以使用 *ping* 通百度服务器 *220.181.38.148*

```
❯ ping baidu.com
PING baidu.com (220.181.38.148) 56(84) bytes of data.
64 bytes from 220.181.38.148 (220.181.38.148): icmp_seq=1 ttl=47 time=25.1 ms
64 bytes from 220.181.38.148 (220.181.38.148): icmp_seq=2 ttl=47 time=24.2 ms
^C
--- baidu.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1179ms
rtt min/avg/max/mdev = 24.220/24.654/25.089/0.434 ms
```

正常访问

只开启 IP 模块，并设置禁止 IP 为百度服务器 ***220.181.38.148***

```
❯ sudo ./test
打开设备
读取设备输出内核信息
no band ping
no band port
no band ip
设置内核
输入xx.xx.xx.xx以禁止IP:220.181.38.148
-1809402404
-1809402404
220.181.38.148
禁止ping(Y/other):
禁止TCP端口(Y/other):
禁止UDP端口(Y/other):
输出内核信息
no band ping
no band port
220.181.38.148
```

禁止该ip响应

禁止 IP 响应

无法 ping 通百度服务器

```
❯ ping baidu.com
PING baidu.com (220.181.38.148) 56(84) bytes of data.
^C
--- baidu.com ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2074ms
```

无法 ping 通百度服务器

查看内核输出如下，响应包被丢弃

```
[ 6089.242653] DROP IP packet from 220.181.38.148
[ 6090.277546] DROP IP packet from 220.181.38.148
[ 6091.321187] DROP IP packet from 220.181.38.148
[ 6117.070803] WSL2: Performing memory compaction.
```

内核输出

4. 端口模块测试

执行命令 ***python3 -m http.server***，在本地 8000 端口运行一个服务器

```
❯ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

运行服务器

使用命令 ***curl 127.0.0.1 8000***，能够正常访问

```
❯ curl 127.0.0.1:8000
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".filter.ko.cmd">.filter.ko.cmd</a></li>
```

正常访问

配置端口模块，并禁止 8000 端口的响应

```
❯ sudo ./test
打开设备
读取设备输出内核信息
no band ping
no band port
no band ip
设置内核
输入xx.xx.xx.xx以禁止IP:
禁止ping(Y/other):
禁止TCP端口(Y/other):y
输入端口:8000
输出内核信息
no band ping
band TCP port 8000
no band ip
```

禁止8000端口

配置端口模块

再次访问 8000 端口，可以发现响应被禁止

```
❯ curl 127.0.0.1:8000
curl: (28) Failed to connect to 127.0.0.1 port 8000: Connection timed out
```

禁止响应

内核输出如下

```
[ 8141.431612] DROP TCP port 8000 IP packet from 127.0.0.1
[ 8142.492199] DROP TCP port 8000 IP packet from 127.0.0.1
[ 8144.572000] DROP TCP port 8000 IP packet from 127.0.0.1
[ 8148.652115] DROP TCP port 8000 IP packet from 127.0.0.1
[ 8156.812018] DROP TCP port 8000 IP packet from 127.0.0.1
```

内核输出

实验体会

通过本次实验，我掌握了 Linux 内核模块编程的方法，进一步了解了 Netfilter 框架的钩子点和钩子处理规则，并熟悉了内核和用户空间的交互方法。

利用钩子函数，我们能够对进出以及转发的数据进行拦截，并针对数据包的内容进行相应的操作，这其实已经有点像一个非常简易防火墙，如果能够对功能做进一步的细化和加强，我们甚至可以利用 Netfilter 提供的钩子实现一个类似于 iptables 在用户空间的强大的防火墙应用程序。