نام و نام خانوادگی : مریم سعیدمهر
ش. د. : ۹۶۲۹۳۷۳
استاد مربوطه : دکتر زالی

پروژه عملی درس سیستم های عامل
دانشگاه صنعتی اصفهان
ترم اول سال تحصیلی ۹۸-۹۹

Packet Filtering Kernel Module Documentation

در ادامه به نحوه ی توسعه و ایجاد یک کرنل ماژول جهت فیلتر کردن پکت ها میپردازیم.

✔ بررسی App_PacketFiltering.c

حاصل کامپایل این کد در user space اجرا میشود و صرفا یک رابط کاربری بین ماژول و کاربر است تا فایل Config.txt که کاربر آن را ایجاد میکند برای تعریف Black List و White List را خوانده و دیتاها را به کرنل برای استفاده ی ماژول بفرستد.

```
int fd = open("/dev/packetfilter", O_RDWR);
```

در این خط ، دیوایس مربوطه را باز میکنیم و سپس کانفیگ را برای آن ارسال میکنیم .

▶ توجه ! خط اول فایل Config.txt مشخص کننده ی نوع کانفیگ است (Black List یا White List)

در ادامه مابقی دیتاهای داخل فایل کانفیگ که به صورت IP:Port است را برای ماژول میفرستیم.

✔ بررسی PacketFilteringKM.c

برای نوشتن لینوکس کرنل ماژول با استفاده از زبان C از کتابخانه های مخصوصی استفاده میکنیم که سیستم عامل برای کدنویسی کرنل در اختیارمان میگذارد.

```
#include <linux/init.h>      // Macros used to mark up functions e.g. __init __exit
#include <linux/module.h>    // Core header for loading LKMs into the kernel
#include <linux/device.h>    // Header to support the kernel Driver Model
#include <linux/kernel.h>    // Contains types, macros, functions for the kernel
#include <linux/fs.h>        // Header for the Linux file system support
#include <linux/uaccess.h>   // Required for the copy to user function
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/skbuff.h>
#include <linux/udp.h>
#include <linux/tcp.h>
#include <linux/icmp.h>
#include <linux/ip.h>
#include <linux/inet.h>
```

کرنل ماژول از دو تابع اصلی **init** و **exit** تشکیل میشود (تابع **main** ندارد!) که **init** فقط در زمان رجیستر شدن ماژول در کرنل فراخوانی و اجرا میشود و تابع **exit** موقع حذف کردن ماژول از کرنل فراخوانی شده و ماژول را **unregister** میکند .

```
module_init(packet_filter_init);
module_exit(packet_filter_exit);
```

تمام تنظیمات لازم برای رجیستر و آنرجیستر کردن ماژول در این دو تابع انجام میشود.

با توجه به اینکه قرار است کانفیگ های ماژول از یک فایل در سطح یوزر به کرنل ارسال شود باید در ماژول یک دیوایس برای ارتباط یوزر با ماژول بسازیم. به علاوه این دیواس باید در تابع **init** ساخته شود و آزاد سازی منابع در تابع **exit** انجام شود.

برای اینکه کاربر بتواند با ماژول تبادل اطلاعات داشته باشد ، باید دیوایسی که ساخته ایم را بشناسد. هر دیوایس برای شناخته شدن دارای دو عدد **major number** و **minor number** است که ترکیب این دو برای هر دیوایس ، یونیک است. بعضی از **major number** ها رزرو شده برای دیوایس های خاص هستند ولی بعضی دیگر هنگام بوت شدن سیستم به دیوایس ها به صورت داینامیکی اختصاص پیدا میکنند. برای **portable** بودن دیوایس مان بهتر است که **major number** مورد نیازش را هنگام ساخت، از سیستم عامل بخواهیم .

```
majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
if (majorNumber<0)
{
    printk(KERN_ALERT "Registering char device failed with %d\n", majorNumber);
    return majorNumber;
}
printk(KERN_INFO "Registering char device has successfully done
                  with major number %d\n", majorNumber);
```

از آنجایی که minor number به نام دیوایس مربوط است ، لزومی ندارد که آن را هم به صورت داینامیک از سیستم عامل بخواهیم.

در این کد fops یک struct از نوع file_operations است که هدرفایل <linux/fs.h> در اختیارمان میگذارد.

```
static struct file_operations fops =
{
  .open = device_open,
  .write = device_write,
  .release = device_release,
};
```

توابع مورد استفاده برای کار با فایل که در استراکچر فوق معرفی شده اند ، در ادامه پیاده سازی شده اند...

```
static int     device_open(struct inode *, struct file *);
static int     device_release(struct inode *, struct file *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);
```

در ادامه ی کد ، دیوایس و کلاس مورد نظر ساخته میشوند :

```
// Register the device class
packetFilterClass = class_create(THIS_MODULE, CLASS_NAME);
if (IS_ERR(packetFilterClass))
{
      // Check for error and clean up if there is
      unregister_chrdev(majorNumber, DEVICE_NAME);
      printk(KERN_ALERT "Failed to register device class\n");
      return PTR_ERR(packetFilterClass);// Correct way to return an error on a pointer
}
printk(KERN_INFO "Device class registered successfully\n");

// Register the device driver
packetFilterDevice = device_create(packetFilterClass,
                                   NULL, MKDEV(majorNumber, 0),
                                   NULL, DEVICE_NAME);
printk(KERN_INFO "Device created successfully with major number %d\n", majorNumber);
if (IS_ERR(packetFilterDevice))
{
      // Clean up if there is an error
      class_destroy(packetFilterClass);
      unregister_chrdev(majorNumber, DEVICE_NAME);
      printk(KERN_ALERT "Failed to create the device\n");
      return PTR_ERR(packetFilterDevice);
}
```

حالا برای انجام رسالت اصلی ماژولمان ، یعنی فیلتر کردن سوکت های عبوری از سیستم مان ، باز هم به یک دیوایس نیاز داریم که در تابع **init** آن را میسازیم . برای این کار هدر <**linux/netfilter.h**> یک استراکچر در اختیارمان میگذارد که از آن استفاده میکنیم ....

```
static struct nf_hook_ops packet_filter __read_mostly =
{
  .pf = NFPROTO_IPV4,
  .priority = NF_IP_PRI_FIRST,
  .hooknum =NF_INET_LOCAL_IN,
  .hook = (nf_hookfn *) packet_hook
};
```

و در ادامه دیوایس نت فیلترمان را میسازیم :

```
packetFilterNet = nf_register_net_hook(&init_net,&packet_filter);
printk(KERN_INFO "Network registered correctly\n");
if(packetFilterNet)
{
     printk(KERN_ALERT"Faild to register a record in netfilter\n");
      class_destroy(packetFilterClass);
      unregister_chrdev(majorNumber, DEVICE_NAME);
      device_destroy(packetFilterClass, MKDEV(majorNumber, 0));// remove the device
      nf_unregister_net_hook(&init_net,&packet_filter);// UnRecord in net filtering
      return packetFilterNet;
}
```

در این استراکچر برای اینکه نشان دهیم فقط سوکت هایی که با پروتکل **ipv4** شناسایی میشوند را برای فیلتر شدن بفرستد ، یعنی سوکت ها در این مرحله ابتدا به این فیلتر می آیند و اگر فیلتر نشدند(یعنی سوکت با پروتکل **ipv4** بودند) برای فیلتر شدن ثانویه که چک کردن **IP:Port** و بررسی مجاز بودن آن است ، فرستاده میشود. در تابع **init** این مورد را تنظیم میکنیم :

```
static struct nf_hook_ops packet_filter __read_mostly =
{
  .pf = NFPROTO_IPV4,
  .priority = NF_IP_PRI_FIRST,
  .hooknum =NF_INET_LOCAL_IN,
  .hook = (nf_hookfn *) packet_hook
};
```

در تابع **packet_hook** در واقع **source IP** و **source Port** پکت ها جدا میشود و براساس **config_mode** و اینکه **IP:Port** در فایل **Config.txt** بوده یا نبوده ، در مورد فیلترشدن یا نشدن آن پکت تصمیم گیری میشود .

در نهایت در تابع **exit** باید کلاس و دیوایس های ساخته شده از بین بروند.....