МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)

Кафедра программного обеспечения вычислительной
техники и автоматизированных систем

Лабораторная работа №4
по дисциплине: «Операционные системы»
на тему: «Разработка драйвера для ОС Linux (Ubuntu)»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверили:
доц. Островский Алексей Мичеславович,
асс. Четвертухин Виктор Романович

Белгород, 2024

Цель работы: Изучить основы разработки драйверов для ядра Linux с использованием языка программирования C, включая настройку окружения, создание драйвера и его тестирование.

Условие индивидуального задания:

Реализовать драйвер для виртуального сетевого устройства, которое принимает входящие пакеты и отправляет обратно тому же отправителю ("эхо").

Ход выполнения работы

Задание 1

Код драйвера:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netdevice.h>
#include <linux/etherdevice.h>

#define DRIVER_NAME "vnet_echo"

static struct net_device *vnet_dev;

static netdev_tx_t vnet_start_xmit(struct sk_buff *skb, struct net_device *dev) {
    if (!skb) {
        printk(KERN_ERR "vnet: Received null skb\\n");
        return NETDEV_TX_OK;
    }

    // Reverse MAC addresses
    struct ethhdr *eth = eth_hdr(skb);
    if (!eth) {
        printk(KERN_ERR "vnet: Failed to get Ethernet header\\n");
        dev_kfree_skb(skb);
        return NETDEV_TX_OK;
    }

    unsigned char tmp_mac[ETH_ALEN];
    memcpy(tmp_mac, eth->h_source, ETH_ALEN);
    memcpy(eth->h_source, eth->h_dest, ETH_ALEN);
```

```
    memcpy(eth->h_dest, tmp_mac, ETH_ALEN);

    // Send the packet back
    skb->dev = dev;

    // Increment statistics
    dev->stats.tx_packets++;
    dev->stats.tx_bytes += skb->len;

    // Send packet
    dev_queue_xmit(skb);
    return NETDEV_TX_OK;
}


static int vnet_open(struct net_device *dev) {
    printk(KERN_INFO "%s: Device opened\n", DRIVER_NAME);
    netif_start_queue(dev);
    return 0;
}

static int vnet_stop(struct net_device *dev) {
    printk(KERN_INFO "%s: Device closed\n", DRIVER_NAME);
    netif_stop_queue(dev);
    return 0;
}

static const struct net_device_ops vnet_netdev_ops = {
    .ndo_open = vnet_open,
    .ndo_stop = vnet_stop,
```

```
    .ndo_start_xmit = vnet_start_xmit,
};

static void vnet_setup(struct net_device *dev) {
    ether_setup(dev);
    dev->netdev_ops = &vnet_netdev_ops;
    dev->flags |= IFF_NOARP;
    dev->features |= NETIF_F_HW_CSUM;
}

static int __init vnet_init(void) {
    printk(KERN_INFO "%s: Initializing the virtual network device\n", DRIVER_NAME);

    vnet_dev = alloc_netdev(0, "vnet%d", NET_NAME_UNKNOWN, vnet_setup);
    if (!vnet_dev) {
        printk(KERN_ERR "%s: Failed to allocate network device\n", DRIVER_NAME);
        return -ENOMEM;
    }

    if (register_netdev(vnet_dev)) {
        printk(KERN_ERR "%s: Failed to register network device\n", DRIVER_NAME);
        free_netdev(vnet_dev);
        return -ENODEV;
    }

    printk(KERN_INFO "%s: Device registered successfully\n", DRIVER_NAME);
    return 0;
```

```
}

static void __exit vnet_exit(void) {
    printk(KERN_INFO "%s: Cleaning up module\n", DRIVER_NAME);
    unregister_netdev(vnet_dev);
    free_netdev(vnet_dev);
}

module_init(vnet_init);
module_exit(vnet_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Ignatiev Artur");
MODULE_DESCRIPTION("Virtual Network Device with Echo Functionality");
```

Описание кода:

1. Создание виртуального устройства: используется alloc_netdev для создания сетевого устройства; настраиваются обработчики операций, такие как ndo_open, ndo_stop, и ndo_start_xmit.
2. Обработка пакетов: в vnet_start_xmit пакеты перехватываются, адреса MAC меняются местами, и пакет отправляется обратно отправителю.
3. Модуль ядра: драйвер регистрируется как модуль ядра и предоставляет необходимые функции инициализации и очистки.

Тестирование

Собираем драйвер:

$ make

Загружаем драйвер в ядро:

$ sudo insmod vnet_echo.ko

Проверяем, что модуль успешно загружен:

$ lsmod | grep vnet_echo

Проверяем наличие нового устройства в списке сетевых интерфейсов:

$ ip link show

Видим интерфейс vnet0.

Присваиваем виртуальному интерфейсу IP-адрес и включите его:

$ sudo ip addr add 192.168.1.100/24 dev vnet0

$ sudo ip link set vnet0 up

Проверяем, что интерфейс активен:

$ ip addr show vnet0

```
user@user-VMware-Virtual-Platform:~/echo_driver$ sudo insmod vnet_echo.ko
user@user-VMware-Virtual-Platform:~/echo_driver$ lsmod | grep vnet_echo
vnet_echo              12288  0
user@user-VMware-Virtual-Platform:~/echo_driver$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
 group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
e DEFAULT group default qlen 1000
    link/ether 00:0c:29:c0:64:b7 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
3: vnet0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
user@user-VMware-Virtual-Platform:~/echo_driver$ sudo ip addr add 192.168.1.100/
24 dev vnet0
user@user-VMware-Virtual-Platform:~/echo_driver$ sudo ip link set vnet0 up
user@user-VMware-Virtual-Platform:~/echo_driver$ ip addr show vnet0
3: vnet0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UNKNOWN group default qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/24 scope global vnet0
       valid_lft forever preferred_lft forever
```

Проверяем функциональность:

Тест с ping:

Проверяем эхо-ответы:

$ ping -I vnet0 192.168.1.100

```
user@user-VMware-Virtual-Platform:~/echo_driver$ ping -I vnet0 192.168.1.100
PING 192.168.1.100 (192.168.1.100) from 192.168.1.100 vnet0: 56(84) bytes of dat
a.
64 bytes from 192.168.1.100: icmp_seq=1 ttl=64 time=0.024 ms
64 bytes from 192.168.1.100: icmp_seq=2 ttl=64 time=0.051 ms
64 bytes from 192.168.1.100: icmp_seq=3 ttl=64 time=0.057 ms
64 bytes from 192.168.1.100: icmp_seq=4 ttl=64 time=0.027 ms
64 bytes from 192.168.1.100: icmp_seq=5 ttl=64 time=0.034 ms
64 bytes from 192.168.1.100: icmp_seq=6 ttl=64 time=0.030 ms
64 bytes from 192.168.1.100: icmp_seq=7 ttl=64 time=0.076 ms
64 bytes from 192.168.1.100: icmp_seq=8 ttl=64 time=0.053 ms
^C
--- 192.168.1.100 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7119ms
rtt min/avg/max/mdev = 0.024/0.044/0.076/0.017 ms
```

Просматриваем пакеты с помощью tcpdump:

$ sudo tcpdump -i vnet0

Видим входящие и исходящие пакеты с интерфейса vnet0.

```
, 1 group record(s)
09:58:06.055119 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0*-
[0q] 2/0/0 (Cache flush) PTR user-VMware-Virtual-Platform.local., (Cache flush)
A 192.168.1.100 (102)
09:58:06.055119 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0 [2
q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
09:58:06.055119 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0*-
[0q] 2/0/0 (Cache flush) PTR user-VMware-Virtual-Platform.local., (Cache flush)
A 192.168.1.100 (102)
09:58:06.055120 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0 [2
q] [2n] ANY (QM)? 100.1.168.192.in-addr.arpa. ANY (QM)? user-VMware-Virtual-Plat
form.local. (114)
09:58:06.055120 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0 [2
q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
09:58:06.055120 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0 [2
q] [2n] ANY (QM)? 100.1.168.192.in-addr.arpa. ANY (QM)? user-VMware-Virtual-Plat
form.local. (114)
09:58:06.055120 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0 [2
q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
09:58:06.055120 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0 [2
q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
09:58:06.055121 IP user-VMware-Virtual-Platform.mdns > mdns.mcast.net.mdns: 0 [2
q] [2n] ANY (QM)? 100.1.168.192.in-addr.arpa. ANY (QM)? user-VMware-Virtual-Plat
form.local. (114)
```

Вывод по работе драйвера виртуального сетевого устройства

Драйвер реализует виртуальное сетевое устройство, которое выполняет эхо-функцию: принимает входящие пакеты и отправляет их обратно отправителю. Основные задачи:

- Создание сетевого интерфейса (vnet0).
- Перехват пакетов, изменение MAC-адресов, чтобы отправить пакет обратно.
- Поддержание минимальной статистики (количество отправленных пакетов и байтов).

Достоинства кода:

- Код использует базовые функции ядра Linux для управления сетевым устройством, такие как alloc_netdev, register_netdev и обработчики событий.
- Драйвер корректно перехватывает пакеты, изменяет заголовки и отправляет их обратно.
- Функции разделены на логические части: vnet_open и vnet_stop для управления состоянием устройства; vnet_start_xmit для обработки пакетов.

Возможные проблемы

1. No buffer space available: эта ошибка возникает, когда драйвер некорректно обрабатывает входящие пакеты, например, не освобождает или неправильно отправляет их. Были добавлены проверки заголовков Ethernet и вызовы освобождения памяти (dev_kfree_skb), чтобы избежать утечек.
2. Код не покрывает тестами, например, обработку невалидных пакетов или большие объёмы данных.
3. Драйвер не фиксирует ошибки при передаче и приёме пакетов.

Варианты улучшений которые расшили бы функционал программы.

1. Сейчас драйвер работает только на уровне Ethernet. Обработка IP- и UDP-заголовков могла бы расширить функциональность устройства.
2. Реализовать счётчики для входящих пакетов (rx_packets) и байтов (rx_bytes).
3. Написать скрипты для автоматической проверки, включая нагрузочные тесты.
4. Проверить использование функции dev_queue_xmit и добавить механизмы очередей для обработки пакетов асинхронно.

Драйвер демонстрирует базовые принципы создания сетевого устройства в Linux. Он подходит для образовательных целей и демонстрации возможностей работы с сетевым стеком ядра. Однако для реального использования драйвер требует доработки.

Вывод: на этой лабораторной работе изучили и на практике выполнили разработку драйвера для ОС Ubuntu на ЯП Си и его тестирование