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.

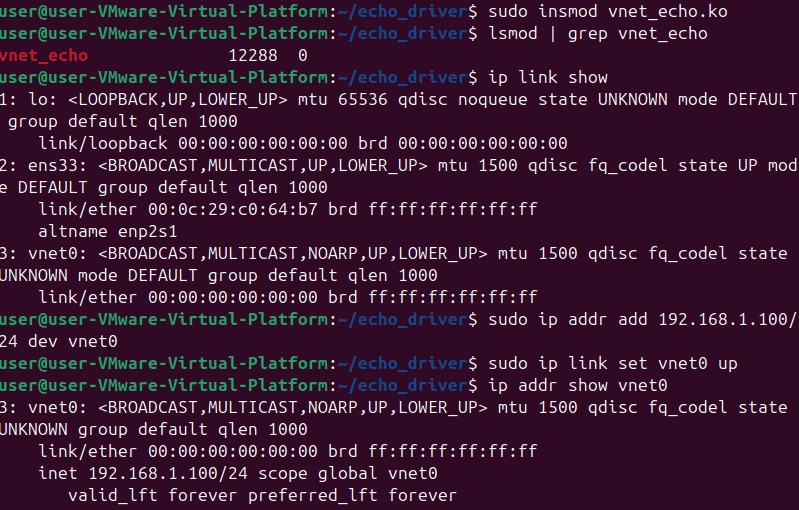
Присваиваем виртуальному интерфейсу IP-адрес и включите его:

$ 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

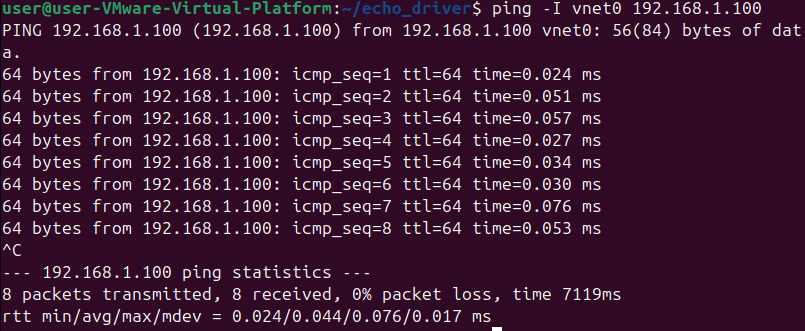


Проверяем функциональность:

Тест с ping:

Проверяем эхо-ответы:

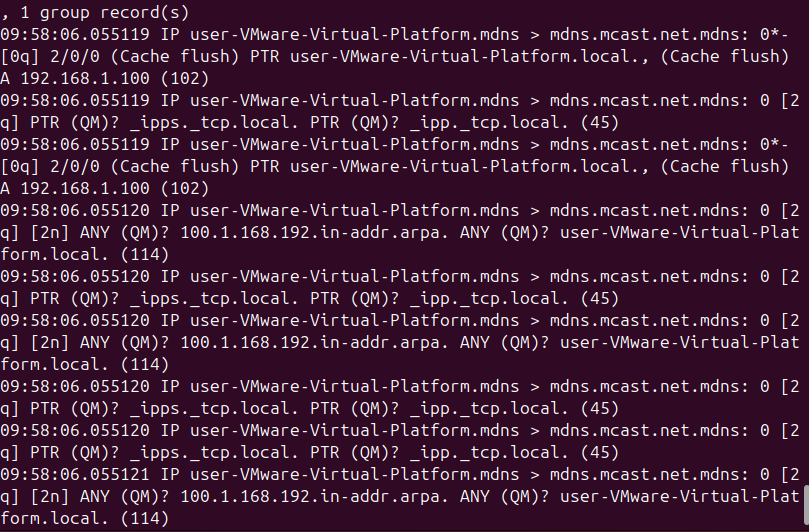
$ ping -I vnet0 192.168.1.100



Просматриваем пакеты с помощью tcpdump:

$ sudo tcpdump -i vnet0

Видим входящие и исходящие пакеты с интерфейса vnet0.



**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 OC Ubuntu на ЯП Си и его тестирование