Сетевой стек OS Course: Дизайн-документ

Ларин Дмитрий

Задача:

1) Система должна поддерживать работу с сетевой картой e1000
2) Поддерживать работу с протоколами Ethernet и IPv4
3) Отвечать на ARP и ICMP запросы
4) Уметь отправлять и принимать UDP пакеты

Реализованный функционал

Взаимодействие с внешними устройствами

В рамках поставленной задачи для операционной системы написаны компоненты для взаимодействия с внешними устройствами. В первую очередь это работа с PCI, а также простейший драйвер для сетевой карты e1000, который отвечает за взаимодействие с эмулированной QEMU сетевой картой e1000

PCI устройство сетевой карты

За основу PCI интерфейса был взят аналогичный из курса MIT (https://pdos.csail.mit.edu/6.828/2016/labs/lab6/). Его детальное описание и описание логики работы можно найти по ссылке.
Фактически, процесс проходит по шине PCI, проверяет подключенные устройства и если они удовлетворяют заданным параметрам (например, подключено устройство e1000) - устройство подключается (pci_attach), затем оно энаблится (pci_func_enable) и данные о нем вносятся в struct pci_struct.

Драйвер e1000

Во время работы кода PCI интерфейса, при наличии в системе информации о подключенном устройстве e1000, происходит вызов e1000_attach. Фактически, оно настраивает MMIO и DMA для доступа к пространству ядра. Таким образом, мы можем получить доступ из ядра к пришедшим на сетевую карту пакетам без копирований (тк адресное пространство сетевой карты отображено на адресное пространство ядра). Более детальное описание так же можно найти в материалах курса MIT: https://pdos.csail.mit.edu/6.828/2016/labs/lab6/
В драйвере реализовано два системных вызова - rx_packet и tx_packet, которые отвечают за получение и отправку сообщений соответственно. Они построены вокруг двух кольцевых буферов. Процессы ядра, обращаясь к этим вызовам, могут перемещать указатели в кольцевых буферах и, таким образом, осуществлять отправку и чтение пакетов из очередей.
Спецификация драйвера e1000: https://pdos.csail.mit.edu/6.828/2021/readings/8254x_GBe_SDM.pdf

Сетевой стек

Поверх PCI и драйвера e1000 был реализован простейший сетевой стек. Он представляет из себя обработчики пакетов, которые, используя системные вызовы драйвера e1000, могут отправлять и получать пакеты с сетевого интерфейса

Общая архитектура

Передача трафика - работает за счет обращения к tx_packet драйвера e1000. Мы создаем пакет, используя описанные далее заголовки, и записываем пакет в выходную очередь.
Получение трафика - для чтения трафика с входного интерфейса используется системный вызов net_serve (kern/net.c). Он привязан к прерываниям от таймера (kern/trap.c). По каждому прерыванию от таймера считывается 2 пакета из входной очереди (при их наличии). Далее происходит разбор трафика - осуществляется проход по всем уровням пришедшего пакета, и, в зависимости от заданной логики, происходит работа с заголовками. Подробнее о реализованной логике - далее.

Прим.: *есть возможность использовать прерывания от сетевой карты в качестве индикатора прихода нового пакета. Для этого необходимо внимательно ознакомиться с соответствующими флагами в спецификации e1000. В рамках данной работы это не было реализовано.*

Обработка Ethernet

Файлы kern/eth.c и kern/eth.h
Реализованы два системных вызова eth_recv и eth_send. Кроме того, задано описание заголовка фрейма Ethernet в struct eth_hdr. Вызов eth_recv используется в net_serve() для получения фреймов из входной очереди, затем, по значению поля eth_type в заголовке фрейма (которое описывает какой протокол находится далее) система приступает к дальнейшему разбору.

Обработка ARP

Файлы kern/arp.c и kern/arp.h
Реализованы два системных вызова arp_recv и handle_arp_request. Описание заголовка находится в kern/arp.h. Вызывается из net_serve, после получения фрейма ethernet с соответствующим типом ETH_TYPE_ARP. Процесс проходит по всем полям заголовка ARP, проверяет его тип. Если тип указан как Request - то пакет передается в handle_arp_request, в котором проверяется наличие нужного адреса в захардкоженной ARP таблице. Если адрес присутствует в таблице, то формируется сообщение ARP Reply и отправляется по адресу из ARP Request с сообщением о наличии нужного адреса на текущей машине

Обработка IPv4

Файлы kern/ip.c и kern/ip.h
Реализованы два системных вызова ip_recv и ip_send. Описание заголовка находится в kern/ip.h. Вызывается из net_serve, после получения фрейма ethernet с соответствующим типом ETH_TYPE_IPV4. Процесс проходит по всем полям заголовка IPv4. Проверяет контрольную сумму. В зависимости от значения поля ip_protocol в заголовке пакета выбирается следующий по стеку протокол (в нашем

случае реализована простейшая поддержка только для протоколов ICMP и UDP). Отправка пакетов при помощи ip_send происходит после формирования заголовка IPv4, а также заголовка Ethernet и выставления в нем типа ETH_TYPE_IPV4

Обработка ICMP

Файлы kern/icmp.c и kern/icmp.h
Реализованы два системных вызова icmp_recv и handle_icmp_request. Описание заголовка находится в kern/icmp.h. Вызывается из функции ip_recv (kern/ip.c) при получении флага IP_PROTO_ICMP. Процесс проходит по всем полям заголовка ICMP, проверяет контрольную сумму и выводит содержимое пакета в консоль. Если был получен пакет ICMP Request, то вызывается функция handle_icmp_request, которая создает пакет ICMP Reply (используя те же идентификаторы и считая новую контрольную сумму) и отправляет его, использую вызов ip_send.

Обработка UDP

Файлы kern/udp.c и kern/udp.h
Реализованы два системных вызова udp_recv и udp_send. Описание заголовка находится в kern/udp.h. Вызывается из функции ip_recv (kern/ip.c) при получении флага IP_PROTO_UDP. Разбирается заголовок пакета и в консоль выводится значение переданных в протоколе данных. Отправка пакета происходит с помощью вызова udp_send - формируется заголовок udp и происходит вызов ip_send.

Тестирование

Тестирование реализовано при помощи дополненного скрипта isp-socket.py (находится в корневой директории проекта). Программа принимает пакеты из сокета QEMU, а также может отправлять в него пакеты. При помощи библиотеки scapy формируется три типа пакетов - UDP, ICMP Request(5 шт) и ARP Request. При корректной работе системы, в консоли ОС нужно увидеть правильно разобранные пакеты. Кроме того, в интерфейсе isp-socket необходимо получить ответные пакеты ARP Reply и ICMP Reply (5 штук).

Также, для более детальной отладки, с QEMU был сделан вывод всех полученных и отправленных в сокет пакетов в файл dump.pcap (создается автоматически при каждом старте QEMU, детали см. в GNUmakefile, флаг QEMUOPTS). Просматривая дамп пакетов можно убедиться в том, что все работает правильно.

Доп: Отладка

Благодаря тому, что физическое устройство e1000 эмулируется QEMU, у нас есть возможность пошагово смотреть как оно работает.
Для драйвера e1000 (находится в hw/net/e1000.c) можно включить E1000_DEBUG. Этот флаг необходимо задефайнить. Затем, пересобрав QEMU, в stderr будут выводится отладочные сообщения эмулированной карты e1000. При необходимости можно поместить вывод отладочных сообщений в нужное место.
Это позволило отлаживать работу драйвера сетевой карты при его разработке и оказалось очень полезным.
Исходный код QEMU и инструкции по сборке можно найти здесь: https://www.qemu.org/download/#source

Доп: Mac OS

На устройствах под управлением Mac OS multicast сокет в QEMU работает некорректно. Для исправления этого необходимо дополнительно пропатчить QEMU.

Патч сделан Виталием Чепцовым:
https://patchew.org/QEMU/20220601164507.51503-1-cheptsov@ispras.ru/