Сеть MINIX3. Часть 3: настройки и виртуализация

Цилюрик О.И.

< olej@front.ru >

Редакция 4.17

от 24.02.2010

Оглавление

Аннотация

Описывается установление виртуальной сети при запуске операционной системы MINIX3 в среде виртуализации QEMU под Linux. После этого рассматриваются вопросы конфигурирования сети для разных целей и топологий, которые не отличается, выполняется ли она для реальной или для виртуальной сетей. Большая часть материала не зависит от вида гостевой (выполняемой в виртуальной машине) операционной системы — MINIX3 взят просто как наименее описанный образец; с равным успехом эти результаты могут быть применены к любой другой POSIX операционной системе.

Введение

Большая часть изложения и примеров будет построена на рассмотрении MINIX3, выполняющегося под управлением системы виртуализации QEMU. Тому есть целый ряд оснований:

– в MINIX3 крайне ограниченный набор поддерживаемых сетевых карт, и даже входящие в него образцы - это устаревшие модели, выходящие из употребления, которые не всегда легко найти.

– виртуальные сети QEMU позволяют смоделировать весьма разветвлённые сетевые конфигурации, которые трудоёмко создать в реальности — на них можно отработать достаточно изощрённые примеры настроек.

– сетевые процессы в виртуальных сетях под QEMU сложнее, а происходящее в реальной сети в точности повторяет их - таким способом проводимое рассмотрения только повышается его степень общности.

Версии системы

Версии MINIX3 в очень большой мере «волатильны» - разработчики часто вносят существенные изменения,

даже не считая должным отражать их даже в man-ах. Вся основная часть описания отрабатывалась на стабильной версии 3.1.6 (релиз 6084), в некоторых случаях, оговоренных особо, рассмотрение ведётся на стабильной версии 3.1.5 (релиз 5612). В используемой вами версии могут быть, порой, довольно существенные отличия, но основные принципы при этом всё равно сохраняются.

Соглашения принятые в тексте

Все показанные в тексте протоколы выполнения команд сохранены прямым копированием с экрана терминала, так же как и графические скриншоты; все действия, описываемые в тексте, могут быть повторно воспроизведены.

В самом тексте, все примеры команд (скопированные с терминала) будет показываться `моноширинным` шрифтом. Кроме того, в большинстве случаев пользовательский ввод в записи команды будет показан `жирным` шрифтом, а ответный вывод от системы — `обычным`. Короткие цитаты из различных источников информации будут показываться *курсивом*.

В конфигурировании сети очень многие действия могут быть выполнены только от имени пользователя `root`. Поскольку оговаривать это в каждом случае невозможно, обратите особое внимание на знак (`$ / #`) приглашения ввода, который показывается в примерах команд: `$` будет означать, что действие может быть выполнено от имени любого пользователя, а # - только от имени суперпользователя `root`.

В дальнейшем рассмотрении мы будем вынуждены не раз прибегать к схемам сетевых топологий, которые будем конфигурировать. Но, во-первых, из нелюбви к рисованию, а, во-вторых, не желая перегружать объём текста множеством рисунков, я буду рисовать сетевые топологии подобно показанному:

```
<minix(MINIX3)>eth0:192.168.3.7<--->tap0:192.168.3.6<-+-><home(Linux)>
                                                       |
              <ADSL>192.168.1.1<--->eth0:192.168.1.7<-+
```

Рис.1

Как понятно из рисунка, для хоста (<...>) мы указываем сетевое имя хоста, и, там где это необходимо, в скобках () уточняем тип операционной системы этого хоста. Перед IP адресом (каждого сетевого интерфейса хоста) может быть указано имя сетевого интерфейса (а может быть и не указано, как для ADSL шлюза в Интернет). При необходимости, после IP может указываться маска, причём записи: `192.168.3.6:255.255.255.0` и `192.168.3.6/24` - будем считать эквивалентными.

Примечание: показанная на рисунке выше (с численными значениями) топология и будет, для определённости, основной в нашем дальнейшем рассмотрении, и именно для неё мы начнём создание виртуальной сети. В таком эталонном фрагменте используется 2 подсети: `192.168.1.0/24` – реальная LAN и `192.168.3.0/24` – виртуальная сеть QEMU.

Установление виртуальной сети

Примечание: если вас не интересуют вопросы установления виртуальных сетей под QEMU, например при работе с реальной сетью в MINIX3, то переходите сразу к следующему разделу о конфигурации сети.

В документации QEMU сказано:

QEMU может эмулировать до 6-и сетевых карт (NE2000-типа). Каждая из карт может быть подключена к определённому сетевому интерфейсу системы-хозяина.

Сеть к виртульной (гостевой) ОС предполагается как виртуальная - мы должны установить в этой виртуальной подсети соединение MINIX3 к базовому Linux, а оттуда уже, по необходимости, обеспечить роутинг в реальную LAN, или в наружу в Интернет.

Предостережение: Во всех описываемых топологиях сети, за исключением одной, где будет оговорено особо, виртуальная подсеть QEMU **не должна совпадать** ни с одной из подсетей, существующих на хосте QEMU. Убедитесь, что создаваемая (как описано далее) виртуальная подсеть не совпадает или перекрывается (по совокупности IP:маска) с диапазонами уже существующих подсетей на Linux хосте – в противном случае вы потеряете множество времени, толкуя весьма странные эффекты в сети. Для установки мы выберем отдельную подсеть `192.168.3.0/24`.

На базовом Linux уже установлена сетевая подсистема, из которой нас будет интересовать:

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:60:52:07:4F:4B
          inet addr:192.168.1.7  Bcast:192.168.1.7  Mask:255.255.255.248
          inet6 addr: fe80::260:52ff:fe07:4f4b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2716 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2875 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1050597 (1.0 MiB)  TX bytes:403440 (393.9 KiB)
          Interrupt:10 Base address:0xe000
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.248 U     0      0        0 eth0
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth0
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
```

- исходная реальная подсеть `192.168.1.0/24`, и её настройка маршрутизации на шлюз `192.168.1.1` во внешний интерфейс.

Виртуальная сеть средствами QEMU

QEMU имеет возможность (определяется параметрами при запуске) создать тунельный интерфейс TAP, и подключать к нему виртуальный сетевой интерфейс запущенной гостевой системы (MINIX3). При таком способе выполняем:

Со стороны Linux:

1. создаём в `/etc` 2 скрипта (старта и останова) для тунельного интерфейса, они имеют права исполнения и установленный SUID бит:

```
# ls -l /etc/qem*
-rwsr-sr-x 1 root root  78 Ноя  8 16:32 /etc/qemu-ifdown
-rwsr-sr-x 1 root root 116 Ноя  8 16:32 /etc/qemu-ifup
# cat /etc/qemu-ifup
#!/bin/sh
echo -------------- tap up -------------
sudo /sbin/ifconfig $1 192.168.2.6
# cat /etc/qemu-ifdown
#!/bin/sh
```

```
echo -------------- tap down -------------
#/sbin/ifconfig $1 down
```

Имена (и полные путевые имена) конфигурационных файлов `qemu-ifup` и `qemu-ifdown` — заданы по умолчанию (для QEMU), при желании их изменить, это можно переопределить в командной строке QEMU. Скрипт остановки в нашем случае реально ничего не делает... , но он может быть, при необходимости, наполнен содержанием.

2. запускаем виртуальную машину так:

```
# qemu -m 100M -hda minix3-disk -boot c -localtime -net nic,vlan=0 -net tap,vlan=0
...
```

- здесь запуск происходит **только** от имени `root`, что связано с необходимостью **создания** и инициализации нового сетевого интерфейса `tap0` (это крайне нежелательно, но как от этого избавиться и возможно ли это, при создании интерфейса средствами QEMU — я не знаю):

```
# ifconfig -a
tap0      Link encap:Ethernet  HWaddr A2:96:61:9C:DB:2A
          inet addr:192.168.3.6  Bcast:192.168.3.255  Mask:255.255.255.0
          inet6 addr: fe80::a096:61ff:fe9c:db2a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:193 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 b)  TX bytes:42362 (41.3 KiB)
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.248 U     0      0        0 eth0
192.168.3.0     *               255.255.255.0   U     0      0        0 tap0
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth0
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
```

Со стороны MINIX3:

1. В `/etc/inet.conf` пропишем сетевой интерфейс `eth0`:

```
eth0 dp8390 0 { default; };
```

- `qemu` устанавливает сетевые интерфейсы типа NE2000, а сетевая плата NE2000 в MINIX3 находится в драйвере `dp8390` (`/usr/src/drivers/dp8390`).

2. Сетевому интерфейсу (умалчиваемому) присвоим IP адрес (после экспериментов это следует поместить куда-то в стартовый скрипт, например, `/etc/rc.net`):

```
# ifconfig -I /dev/ip0
ifconfig: /dev/ip0: Host address not set
# ifconfig -I /dev/ip0 -h 192.168.3.7
# ifconfig -av
/dev/ip0: address 192.168.3.7 mtu 1500
```

Простейшая сеть создана! После этого мы можем уже пинговать достижимость достижимость гостевого MINIX3 со стороны хоста базового Linux:

```
$ ping 192.168.3.7
PING 192.168.3.7 (192.168.3.7) 56(84) bytes of data.
64 bytes from 192.168.3.7: icmp_seq=1 ttl=96 time=169 ms
64 bytes from 192.168.3.7: icmp_seq=2 ttl=96 time=58.8 ms
64 bytes from 192.168.3.7: icmp_seq=3 ttl=96 time=59.5 ms
--- 192.168.3.7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 58.882/95.945/169.399/51.940 ms
```

Или, напротив, достижимость Linux интерфейса `tap0` со стороны MINIX3[1]:

```
$ ping 192.168.3.6
192.168.3.6 is alive
```

На этом установление виртуальной сети `192.168.3.0/24` можем считать законченным, но она ещё требует разнообразных настроек маршрутизации, к чему мы вернёмся в следующем разделе о конфигурировании сети.

Управление тунельным интерфейсом

В предыдущем примере мы полагались на умение QEMU при запуске создавать тунельный интерфейс. Но создавать тунельный интерфейс и управлять его свойствами мы можем и более гибко, и самостоятельно. Для этого нам понадобится, обычно отсутствующая в дистрибутивах Linux, программа `tunctl` из проекта: http://sourceforge.net/projects/tunctl/files/tunctl/1.5/tunctl-1.5.tar.gz/download .

Установка:

```
$ pwd
/usr/src/tunctl-1.5
$ make
cc -g -Wall -o tunctl tunctl.c
docbook2man tunctl.sgml
Using catalogs: /etc/sgml/sgml-docbook-4.1-1.0-30.1.cat
Using stylesheet: /usr/share/sgml/docbook/utils-0.6.14/docbook-utils.dsl#print
Working on: /usr/src/tunctl-1.5/tunctl.sgml
<mdz@debian.org>
MattZimmerman2001Matt ZimmermanJuly  9, 2008Done.
# make install
install -d /usr/sbin
install tunctl /usr/sbin
install -d /usr/share/man/man8
install tunctl.8 /usr/share/man/man8
# which tunctl
/usr/sbin/tunctl
$ man tunctl
```

1 С `ping` в MINIX3 нужно быть осторожнее, о чём мы поговорим подробнее позже.

```
...
   To create an interface for use by a particular user, invoke tunctl without
   the -d option:
          # tunctl -u someuser
   Set   tap0   persistent and owned by   someuser
   Then, configure the interface as normal:
          # ifconfig tap0 192.168.0.254 up
          # route add -host 192.168.0.253 dev tap0
          # bash -c   echo 1 > /proc/sys/net/ipv4/conf/tap0/proxy_arp
          # arp -Ds 192.168.0.253 eth0 pub
...
   To delete the interface, use the -d option:
          # tunctl -d tap0
   Set   tap0   nonpersistent
```

Для работы `tunctl` необходимо наличие в Linux модуля ядра:

```
[root@home aqemu]# lsmod | grep tun
tun                    14529  0
```

Но обычно, даже не в самых свежих версиях Linux, это не составляет проблемы, этот модуль присуствует, и мы не станем на этом останавливаться.

Создадим новый тунельный интерфейс не пользуясь услугами QEMU. Но для начала проверим отсутствие такого интерфейса:

```
[root@home ~]# ip link show tap
Device "tap" does not exist.
```

Создаём и инициализируем интерфейс:

```
[root@home ~]# tunctl -u olej -t tap0
Set 'tap0' persistent and owned by uid 500
[root@home ~]# ifconfig tap0 192.168.3.6/24
[root@home ~]# ifconfig tap0 up
[root@home ~]# ip addres show tap0
6: tap0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 500
    link/ether 2a:2d:99:a8:a4:0c brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.6/24 brd 192.168.3.255 scope global tap0
    inet6 fe80::282d:99ff:fea8:a40c/64 scope link
       valid_lft forever preferred_lft forever
```

Интерфейс готов. Некоторые подготовительные операции к запуску QEMU (сейчас станет понятно зачем):

```
[root@home ~]# cat /proc/sys/net/ipv4/conf/tap0/proxy_arp
0
[root@home ~]# echo 1 > /proc/sys/net/ipv4/conf/tap0/proxy_arp
[root@home ~]# cat /proc/sys/net/ipv4/conf/tap0/proxy_arp
1
[root@home ~]# ls -l /dev/net/tun
```

```
crw------- 1 root root 10, 200 Фев 22 03:27 /dev/net/tun
[root@home ~]# chmod a+rw /dev/net/tun
```

Теперь мы можем произвести запуск QEMU — обратите внимание, что на этот раз мы запускаем виртуальную машину с поддержкой интерфейса `tap0`, но не от имени `root`, а от имени ординарного пользователя (это первое отличие, которого мы добивались, а о втором — чуть позже):

```
[olej@home win_e]$ qemu -m 70M -hda minix3-r6084-800 -boot c \
-kernel-kqemu -localtime -net nic,vlan=0 -net tap,vlan=0,ifname=tap0,script=no
...
```

Убеждаемся в том, что сеть `192.168.3.0` работоспособна:

```
[olej@home ~]$ ping 192.168.3.7
PING 192.168.3.7 (192.168.3.7) 56(84) bytes of data.
64 bytes from 192.168.3.7: icmp_seq=1 ttl=96 time=184 ms
64 bytes from 192.168.3.7: icmp_seq=2 ttl=96 time=54.4 ms
64 bytes from 192.168.3.7: icmp_seq=3 ttl=96 time=53.7 ms
...
[olej@home ~]$ telnet 192.168.3.7
Trying 192.168.3.7...
Connected to minix (192.168.3.7).
Escape character is '^]'.
Minix  Release 3 Version 1.6  (ttyp0)
minix login: root
Password:
...
Terminal type? (network) minix
# uname -a
Minix minix 3 1.6 i686
# shutdown now
Broadcast message from root@minix (ttyp0)
Mon Feb 22 04:38:14 2010...
```

Командой `shutdown` мы завершили выполнение MINIX3 (и выполняющий его QEMU). А теперь смотрим на интерфейс `tap0`:

```
[root@home aqemu]# ifconfig tap0
tap0      Link encap:Ethernet  HWaddr EE:34:22:47:67:2C
          inet addr:192.168.3.6  Bcast:192.168.3.255  Mask:255.255.255.0
          inet6 addr: fe80::ec34:22ff:fe47:672c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:113 errors:0 dropped:0 overruns:0 frame:0
          TX packets:120 errors:0 dropped:110 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:7516 (7.3 KiB)  TX bytes:8271 (8.0 KiB)
```

В этом и заключается вторая особенность, которой мы добивались: интерефейс `tap0` существует независимо от запуска QEMU, и к нему может подключаться более одной виртуальной машины QEMU.

И наконец, когда у нас исчезнет необходимость в нём, мы можем удалить интерфейс так же легко, как мы его и создали:

```
[root@home aqemu]# tunctl -d tap0
Set 'tap0' nonpersistent
[root@home aqemu]# ip link show tap0
Device "tap0" does not exist.
```

Ещё более развёрнутый и сложный пример управления тунельным интерфейсом мы рассмотрим позже, при рассмотрении сетевых мостов.

Конфигурации сети

Сетевые определения

Несколько файлов (все они находятся в `/etc`) содержат определения констант, используемых сетевыми средствами. Файл `/etc/protocols` содержит численные определения для транспортных протоколов IP, а `/etc/services` - численные определения для протоколов прикладного уровня. Образцы записей в этих файлах:

```
# cat /etc/protocols
#
# Internet (IP) protocols
#
#       @(#)protocols   8.1 (Berkeley) 6/9/93
#
ip      0       IP              # internet protocol, pseudo protocol number
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group management protocol
ggp     3       GGP             # gateway-gateway protocol
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
hmp     20      HMP             # host monitoring protocol
xns-idp 22      XNS-IDP         # Xerox NS IDP
rdp     27      RDP             # reliable data protocol
iso-tp4 29      ISO-TP4         # ISO Transport Protocol Class 4
iso-ip  80      ISO-IP          # ISO Internet Protocol
encap   98      ENCAP           # RFC1241 encapsulation
# head -n 20 /etc/services
#
# Network services, Internet style
#
#       @(#)services    8.1 (Berkeley) 6/9/93
#
tcpmux          1/tcp                           # TCP port multiplexer (RFC1078)
echo            7/tcp
```

```
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp             21/tcp
ssh             22/tcp    #Secure Shell Login
ssh             22/udp    #Secure Shell Login
...
```

Конфигурирационные файлы сети

Ниже описываются те файлы (все они находятся в `/etc`), которые изменяются в ходе конкретных сетевых настроек данного хоста.

Файл `/etc/inet.conf` содержит определения сетевых интерфейсов хоста, чаще всего, при одном сетевом интерфейсе, он имеет вид подобный следующему:

```
# cat inet.conf
eth0 dp8390 0 { default; } ;
```

При этом создаются определения сетевых интерфейсов:

```
# ls -l /dev/*eth* /dev/*ip*
crw------- 2 root  operator  7,   1 Feb 11 16:09 /dev/eth
crw------- 2 root  operator  7,   1 Feb 11 16:09 /dev/eth0
crw------- 2 root  operator  7,   2 Feb 11 16:09 /dev/ip
crw------- 2 root  operator  7,   2 Feb 11 16:09 /dev/ip0
crw-rw-rw- 1 root  operator  7,   0 Feb 11 16:09 /dev/ipstat
```

В файл `/etc/inet.conf` могут добавляться записи (строки) для каждого сетевого интерфейса. При использовании, например, интерфейса последовательного порта RS-232:

```
# cat /etc/inet.conf
eth0 dp8390 0 ;
psip1 { default; } ;
```

При этом добавится и число интерфейсов в `/dev` :

```
# ls /dev/*ip*
/dev/ip   /dev/ip0   /dev/ip1   /dev/ipstat   /dev/psip   /dev/psip1
```

Файл `/etc/rc.net` содержит команды, которые обычно определяет IP-привязку определённых ранее (в `/etc/inet.conf`) интерфейсов к IP, роутинг и другие сетевые особенности. Команды из файла будут выполнены `/etc/rc.net` однократно при загрузке системы, при выполнении стартового скрипта `/etc/rc`. Обычно содержимое `/etc/rc.net` подобно следующему:

```
# cat /etc/rc.net
ifconfig -I /dev/ip0 -n 255.255.255.0 -h 192.168.3.7
```

```
add_route -g 192.168.3.7
daemonize nonamed -L
```

Такое содержимое обычно прописывается в ходе диалога при начальной установке MINIX3. Сюда же можно поместить команды старта сетевых серверов: `telnet, ftp, rlogin`...

Примечание: Как будет показано далее, определение роутинга подобное показанному (генерируемое системой), годится для реальной сети, но не годится для виртуальной под QEMU.

Файл `/etc/hostname.file` определяет сетевое имя этого хоста (оно же выводится в текстовой консоли в подсказке на вход):

```
# cat hostname.file
minix
```

Файл `/etc/hosts`, как и во всякой POSIX системе, определяет перечень известных хостов, для простейшего разрешения сетевых имён в IP адреса. Его содержание будет рассмотрено далее подробно.

Порядок инициализации стартовых скриптов

При загрузке системы однократно выполняется следующая последовательность основных стартовых скриптов (порядок вызовов легко проследить из самого текста скриптов):

```
/etc/rc -> /usr/etc/rc -> /usr/etc/rc.local -> /etc/rc.net
```

При **каждом входе** (т.е. это может повторяться многократно) пользователя (`login` в консоли, `telnet` подключение) выполняется такая последовательность скриптов:

```
/etc/profile -> $HOME/.profile -> $HOME/.ashrc
```

Последний выполняющийся скрипт — это инициализация экземпляра используемой оболочки shell, `$HOME/.ashrc` записан по умолчанию, при использовании, например, `bash` это будет `$HOME/.bashrc`.

Зная и учитывая эти последовательности, мы можем прописать в них в нужных местах собственные команды управления сетью (примеры которых мы рассматриваем дальше), для того, чтобы требуемая конфигурация устанавливалась после загрузки.

Интерфейсы

Проверяем конфигурации аппаратных сетевых интерфейсов в `/etc/inet.conf`. Например, при использовании сетевой карты и 2-х последовательных портов, это может быть:

```
# cat /etc/inet.conf
eth0 dp8390 0 { default; } ;
psip1 ;
psip2 ;
```

Примечание: При использовании RS-232 в качестве каналов передачи данных, а не терминальных линий, нужно обязательно закомментировать соответствующие строки в `/etc/ttytab` :

```
# cat /etc/ttytab | head -n 10
# ttytab - terminals
#
# Device        Type            Program         Init
console         minix           getty
```

```
ttyc1           minix           getty
ttyc2           minix           getty
ttyc3           minix           getty
#tty00          unknown
#tty01          unknown
ttyp0           network
...
```

Каждому физическому интерфейсу, используемому системой, должна соответствовать одна строчка в файле `/etc/inet.conf`.

Определения хостов

Речь идёт, как вы поняли, о файле `/etc/hosts`. В нём записаны IP часто используемых хостов, и демон `nonamed` использует его для разрешения имён. В самом MINIX3 нет другого (DNS) механизма разрешения имён (типа `bind`), но система может пользоваться внешними DNS, как будет показано дальше. Для определённости примеров далее я привожу свой `/etc/hosts`, который буду использовать в дальнейшем изложении.

```
# cat /etc/hosts
192.168.3.7     %nameserver     #minix
64.102.255.44   %nameserver     #DNS 1
128.107.241.185 %nameserver     #DNS 2


192.168.3.7     minix
72.249.144.181  qnx.org.ru
```

Маршрутизация на базовый хост

Посмотрим поведение показанного на рисунке фрагмента топологии (показанной ранее на рис.1) при созданных выше настройках:

```
<minix(MINIX3)>eth0:192.168.3.7<--->tap0:192.168.3.6<-+-><home(Linux)>
                                                      |
                                        eth0:192.168.1.7<-+
```

Рис.2

В этой конфигурации:

```
# ifconfig -av
/dev/ip0: address 192.168.3.7 netmask 255.255.255.0 mtu 1500
# pr_routes
ent #   if              dest         gateway dist  pref  mtu flags
    0  ip0         0.0.0.0/0      192.168.3.7    1     0    0 static
```

Для отслеживания того, что происходит, запускаем по экземпляру программы `tcpdump` и на Linux и на MINIX3 хостах, и смотрим только ICMP пакеты (запуски `tcpdump` будут показаны дальше, вместе с результатами). И выполняем несколько команд на MINIX3:

```
# ping 192.168.3.7
192.168.3.7 is alive
# ping 192.168.3.6
192.168.3.6 is alive
# ping 192.168.1.7
```

```
192.168.1.7 is alive
# ping qnx.org.ru
qnx.org.ru is alive
# ping 69.70.20.198
69.70.20.198 is alive
```

Всё нормально? В лучшем случае вы увидите на 1-й консоли MINIX3 сообщение:

```
# tail -n -1 /usr/log/messages
Feb 21 05:54:04 minix kernel: ip[0]: no route to 69.70.20.198
```

А вот то, что видели `tcpdump` (или, точнее, чего не увидели) на сетевых интерфейсах:

В Linux:

```
$ sudo /usr/sbin/tcpdump -n -i any ip proto \\icmp
tcpdump: WARNING: Promiscuous mode not supported on the "any" device
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 96 bytes
07:26:27.379208 IP 192.168.3.7 > 192.168.3.6: ICMP echo request, id 0, seq 0, length 10
07:26:27.379425 IP 192.168.3.6 > 192.168.3.7: ICMP echo reply, id 0, seq 0, length 10
```

В MINIX3:

```
# /usr/local/sbin/tcpdump -D
1.eth0
# /usr/local/sbin/tcpdump -i eth0 ip proto \\icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 68 bytes
05:25:34.116666 IP minix > 192.168.3.6: ICMP echo request, id 0, seq 0, length 10
05:25:35.050000 IP 192.168.3.6 > minix: ICMP echo reply, id 0, seq 0, length 10
^Ctcpdump: pcap_loop: read: Interrupted system call
2 packets captured
0 packets received by filter
0 packets dropped by kernel
```

Только к одному IP был произведен ICMP запрос: `192.168.3.6` . К остальным IP `ping` MINIX3 даже не пытался произвести запрос, но ответил о достижимости хоста! И только `tcpdump` позволил нам обнаружить это. Маршрутизация пакетов из QEMU MINIX3 на базовый Linux хост не происходит. Для установления маршрутизации с QEMU MINIX3 на базовый хост Linux, файл `/etc/rc.net` нужно изменить. Например так:

```
# cat /etc/rc.net
ifconfig -I /dev/ip0 -n 255.255.255.0 -h 192.168.3.7
add_route -g 192.168.3.7 -d 192.168.3.6 -n 255.255.255.255 -I /dev/ip0
add_route -g 192.168.3.6 -d 0.0.0.0 -n 0.0.0.0
daemonize nonamed -L
/usr/bin/tcpd telnet /usr/bin/in.telnetd &
/usr/bin/tcpd ftp /usr/bin/in.ftpd &
/usr/bin/tcpd login /usr/bin/in.rlogind &
```

Здесь добавлен и запуск серверов `telnetd, ftpd, rlogind` для удалённого доступа к MINIX3 хосту.

Динамически, без перезагрузки MINIX3, мы добиваемся того же результата (в маршрутизации) последовательностью команд:

```
# del_route -g 192.168.3.7 -I /dev/ip0
# add_route -g 192.168.3.7 -d 192.168.3.6 -n 255.255.255.255 -I /dev/ip0
# add_route -g 192.168.3.6 -d 0.0.0.0 -n 0.0.0.0
# pr_routes
ent #   if              dest         gateway dist  pref  mtu flags
    0  ip0     192.168.3.6/32     192.168.3.7    1     0    0 static
    1  ip0         0.0.0.0/0      192.168.3.6    1     0    0 static
```

Повторяем проверки на доступность (только то, что не получалось раньше):

```
# ping 192.168.1.7
192.168.1.7 is alive
# ping 192.168.1.1
no answer from 192.168.1.1
# ping qnx.org.ru
no answer from qnx.org.ru
```

Если проследить `tcpdump` на Linux стороне (разделитель пустой строкой добавлен мной после каждого `ping` для наглядности):

```
$ sudo /usr/sbin/tcpdump -n -i any ip proto \\icmp
06:46:11.597702 IP 192.168.3.7 > 192.168.1.7: ICMP echo request, id 0, seq 0, length 10
06:46:11.597963 IP 192.168.1.7 > 192.168.3.7: ICMP echo reply, id 0, seq 0, length 10

06:46:22.783473 IP 192.168.3.7 > 192.168.1.1: ICMP echo request, id 0, seq 0, length 10
06:46:22.783647 IP 192.168.3.7 > 192.168.1.1: ICMP echo request, id 0, seq 0, length 10
... и так 40 запросов ...

06:47:14.347858 IP 192.168.3.7 > 72.249.144.181: ICMP echo request, id 0, seq 0, length 10
06:47:14.348037 IP 192.168.3.7 > 72.249.144.181: ICMP echo request, id 0, seq 0, length 10
... и так 40 запросов ...
```

Теперь запросы к базовому хосту QEMU/Linux доходят благополучно. Но запросы ICMP к другим хостам уходят сквозь базовый хост Linux, но не возвращаются назад ни из LAN, ни из внешнего Интернет. Это уже вопросы маршрутизации, и этим мы займёмся позже.

Примечание: Да! Не забудьте включить и проверить форвардинг IP пакетов сквозь базовый хост:

```
[root@home ~]# cat /proc/sys/net/ipv4/ip_forward
0
[root@home ~]# echo 1 > /proc/sys/net/ipv4/ip_forward
[root@home ~]# cat /proc/sys/net/ipv4/ip_forward
1
```

Проверяем маршрутизацию между виртуальным MINIX3 и базовым Linux, например так:

```
# ssh -l root 192.168.1.7
```

```
root@192.168.1.7's password:
Last login: Sun Nov 29 18:14:20 2009 from 192.168.6.4
[root@home ~]# uname -a
Linux home 2.6.18-92.el5 #1 SMP Tue Jun 10 18:49:47 EDT 2008 i686 i686 i386 GNU/Linux
...
```

Теперь всё по честному!

Маршрутизация в LAN

Посмотрим такой фрагмент LAN:

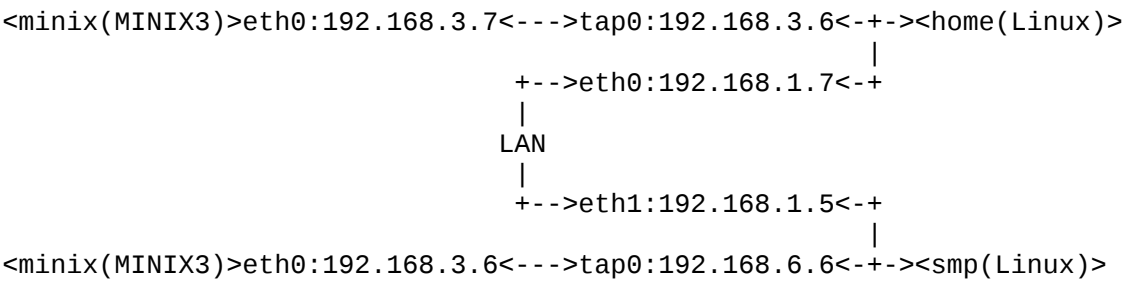

Рис.3

Сейчас в LAN запущено 2 виртуальных MINIX3, на разных базовых хостах Linux. Конечно, в этом случае виртуальные подсети каждого из MINIX3 хостов должны различаться (выбраны `192.168.3.3/24` и `192.168.3.6/24`). В таком случае можно прописать жёсткие статические схемы маршрутизации на Linux хостах. Этот способ грубоват, но не требует привлечения никаких иных средств и механизмов, сверх базовых TCP/UP.

На хосте `smp` :

```
[root@smp ~]# route -v add -net 192.168.3.0 netmask 255.255.255.0 gw 192.168.1.7 dev eth1
[root@smp ~]# route -ve
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.6.0     *               255.255.255.0   U         0 0          0 tap0
192.168.3.0     home            255.255.255.0   UG        0 0          0 eth1
192.168.1.0     *               255.255.255.0   U         0 0          0 eth1
169.254.0.0     *               255.255.0.0     U         0 0          0 eth1
default         192.168.1.1     0.0.0.0         UG        0 0          0 eth1
```

На другом хосте, `home` :

```
[root@home ~]# route -v add -net 192.168.6.0 netmask 255.255.255.0 gw \
               192.168.1.5 dev eth0
[root@home ~]# route -ve
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.1.0     *               255.255.255.248 U         0 0          0 eth0
192.168.6.0     smp             255.255.255.0   UG        0 0          0 eth0
192.168.3.0     *               255.255.255.0   U         0 0          0 tap0
```

```
192.168.122.0   *               255.255.255.0   U         0 0          0 virbr0
169.254.0.0     *               255.255.0.0     U         0 0          0 eth0
default         192.168.1.1     0.0.0.0         UG        0 0          0 eth0
```

Теперь можно с Linux хоста (`smp`) проверить доступность удалённого виртуального MINIX3 (работающего на другом Linux хосте):

```
[root@smp ~]# ping 192.168.3.6
PING 192.168.3.6 (192.168.3.6) 56(84) bytes of data.
64 bytes from 192.168.3.6: icmp_seq=1 ttl=64 time=1.62 ms
64 bytes from 192.168.3.6: icmp_seq=2 ttl=64 time=0.396 ms
```

А вот как выглядит ping с одного из MINIX3 хоста к поочерёдно всем промежуточным точкам трассы до другого MINIX3 хоста, начиная с него же самого:

```
# ping 192.168.3.7
192.168.3.7 is alive
# ping 192.168.3.6
192.168.3.6 is alive
# ping 192.168.1.7
192.168.1.7 is alive
# ping 192.168.1.5
192.168.1.5 is alive
# ping 192.168.6.6
192.168.6.6 is alive
# ping 192.168.6.3
192.168.6.3 is alive
```

Как выйти во внешнюю сеть?

IP маскарадинг (NAT)

В таком варианте мы будем на интерфейсе `eth0` (связь с физической LAN) подменять исходящий IP для пакетов исходящих из сети `192.168.3.0/24`. Начнём (чтоб не повторять всё изначально) с сети в том состоянии, как мы оставили её в предыдущих главах:

```
[olej@home ~]$ ping 192.168.3.7
PING 192.168.3.7 (192.168.3.7) 56(84) bytes of data.
64 bytes from 192.168.3.7: icmp_seq=1 ttl=96 time=202 ms
64 bytes from 192.168.3.7: icmp_seq=2 ttl=96 time=54.1 ms
...
```

Добавляем правило преобразования IP:

```
[root@home ~]# iptables -t nat -A POSTROUTING -s 192.168.3.0/24 -j MASQUERADE
[root@home ~]# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

```
MASQUERADE  all  --  192.168.122.0/24    !192.168.122.0/24
MASQUERADE  all  --  192.168.3.0/24       anywhere
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Запустим на Linux хосте программу (фрагменты её вывода я буду показывать далее):

```
[olej@home ~]$ sudo /usr/sbin/tcpdump -n -i any ip proto \\icmp
```

А на хосте MINIX3 будем последовательно выполнять (после каждой команды показан фрагмент порождённого ею вывода `tcpdump`):

```
# ping 192.168.3.6
192.168.3.6 is alive

05:28:41.941237 IP 192.168.3.7 > 192.168.3.6: ICMP echo request, id 0, seq 0, length 10
05:28:41.941414 IP 192.168.3.6 > 192.168.3.7: ICMP echo reply, id 0, seq 0, length 10

# ping 192.168.1.7
192.168.1.7 is alive

05:28:45.731972 IP 192.168.3.7 > 192.168.1.7: ICMP echo request, id 0, seq 0, length 10
05:28:45.732184 IP 192.168.1.7 > 192.168.3.7: ICMP echo reply, id 0, seq 0, length 10

# ping qnx.org.ru
qnx.org.ru is alive

05:29:22.922808 IP 192.168.3.7 > 72.249.144.181: ICMP echo request, id 0, seq 0, length 10
05:29:22.923000 IP 192.168.1.7 > 72.249.144.181: ICMP echo request, id 0, seq 0, length 10
05:29:23.138341 IP 72.249.144.181 > 192.168.1.7: ICMP echo reply, id 0, seq 0, length 10
05:29:23.138524 IP 72.249.144.181 > 192.168.3.7: ICMP echo reply, id 0, seq 0, length 10

# ping 64.102.255.44
64.102.255.44 is alive

06:18:46.521976 IP 192.168.3.7 > 64.102.255.44: ICMP echo request, id 0, seq 0, length 10
06:18:46.522166 IP 192.168.1.7 > 64.102.255.44: ICMP echo request, id 0, seq 0, length 10
06:18:46.709080 IP 64.102.255.44 > 192.168.1.7: ICMP echo reply, id 0, seq 0, length 26
06:18:46.709233 IP 64.102.255.44 > 192.168.3.7: ICMP echo reply, id 0, seq 0, length 26
```

Хорошо видно, что ICMP запросы к интерфейсам базового хоста QEMU непосредственно порождают ICMP ответы (первые 2 `ping`). Но как только ICMP пакет направляется либо в физическую LAN, либо через шлюз этой LAN во внешние сети — то здесь каждый ICMP пакет из сети `192.168.3.0/24` тут же дублируется ICMP пакетом с исходящим адресом интерфейса `192.168.1.7`.

В качестве дополнительного подтверждения доступности Интернет, проверяем разрешение имени из MINIX3 виртуального хоста посредством внешнего, размещённого где-то в Интернет, DNS сервера:

```
# host -v qnxclub.net 64.102.255.44
Using domain server 64.102.255.44:
Trying domain ""
```

```
rcode = 0 (Success), ancount=1
The following answer is not authoritative:
qnxclub.net     86400 IN        A       69.70.20.198
For authoritative answers, see:
qnxclub.net     86400 IN        NS      dns2.is47.com
qnxclub.net     86400 IN        NS      dns1.is47.com
Additional information:
dns1.is47.com   172799 IN       A       69.70.20.197
dns2.is47.com   172799 IN       A       69.70.20.196
```

Обратно тому, как мы устанавливали правило преобразования IP адресов, мы его можем и убрать:

```
[root@home ~]# iptables -t nat -D POSTROUTING -s 192.168.3.0/24 -j MASQUERADE
[root@home ~]# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  192.168.122.0/24    !192.168.122.0/24
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Примечание: обсуждения и документация по `iptables` (см. в конце текста) рекомендуют указывать действие SNAT, а не MASQUERADE всегда, когда это возможно, при этом нагрузка на процессор будет значительно ниже. И в данном случае это работает! Тогда трансляцию NAT мы устанавливаем командой:

```
[root@home ~]# iptables -t nat -A POSTROUTING -s 192.168.3.0/24 \
-j SNAT --to-source 192.168.1.7
[root@home ~]# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  192.168.122.0/24    !192.168.122.0/24
SNAT       all  --  192.168.3.0/24       anywhere             to:192.168.1.7

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

А удаляем трансляцию, соответственно:

```
[root@home ~]# iptables -t nat -A POSTROUTING -s 192.168.3.0/24 \
-j SNAT --to-source 192.168.1.7
```

Сетевой мост

Совершенно отличающийся подход от показанного в предыдущем разделе – использование в Linux программного сетевого моста (команда `brctl`) объединяющего два разнородных сегмента (интерфейсы `eth0` и `tap0`) на уровне MAC пакетов, соединяющего их в единую IP сеть. Здесь уже не приходится говорить о 2-х IP подсетях (`192.168.1.0` и `192.168.3.0`), и примеры, чтобы не возбуждать ненужные ассоциации, я буду

показывать в другой настройке LAN – 192.168.2.0/24. С таким вот интерфейсом хоста в физическую LAN:

```
[root@opos9 ~]# ip addres show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:d0:b7:16:9f:0d brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.108/24 brd 192.168.2.255 scope global eth0
    inet6 fe80::2d0:b7ff:fe16:9f0d/64 scope link
       valid_lft forever preferred_lft forever
```

Создаём сетевой мост:

```
[root@opos9 etc]# brctl addbr br0
[root@opos9 ~]# brctl addbr br0
[root@opos9 ~]# ip addres show br0
5: br0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
```

- на пока, как легко видеть, у интерфейса моста нет не только IP, на даже MAC адреса, но сам интерфейс уже есть.

Теперь нам нужно остановить реальный Ethernet интерфейс eth0, и перевести его с режим promiscuous (режим прослушивания, неразборчивый режим – существуют разные синонимы), в котором оно получает весь трафик, приходящий на интерфейс; в ином состоянии утилита brctl просто откажется подключать интерфейс к мосту. После этого мы, снова запустив его, подключим интерфейс к мосту. А мост конфигурируем на тот IP адрес, на который был раньше конфигурирован физический интерфейс eth0:

```
[root@opos9 ~]# ifconfig eth0 down
[root@opos9 ~]# ifconfig eth0 0.0.0.0 promisc
[root@opos9 ~]# ifconfig eth0 up
[root@opos9 ~]# brctl addif br0 eth0
[root@opos9 ~]# ifconfig br0 192.168.2.108
```

Предупреждение: между отключением физического интерфейса (1-я команда этой группы) и установкой IP для моста (последняя команда) вы теряете связь хоста с LAN. Поэтому: а). перед выполнением этой группы команд тщательно продумайте свои действия, чтобы всё корректно восстановить, и б). после выполнения проверьте восстановление связи с LAN:

```
[root@opos9 ~]# ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=255 time=2004 ms
64 bytes from 192.168.2.1: icmp_seq=2 ttl=255 time=1005 ms
```

Но это ещё не значит, что у вас восстановилась связь далее шлюза 192.168.2.1! Потому, что:

```
[root@opos9 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 br0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

- у вас нет шлюза по умолчанию, который ранее был назначен на интерфейс eth0. Восстановим умалчиваемый шлюз.

```
[root@opos9 ~]# route add default gw 192.168.2.1 dev br0
```

```
[root@opos9 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 br0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
0.0.0.0         192.168.2.1     0.0.0.0         UG    0      0        0 br0
```

Вот теперь у вас восстановлен Интернет, но трафик идёт не через интерфейс `eth0`, а через мост `br0`. Смотрим состояния интерфейсов:

```
[root@opos9 ~]# brctl show br0
bridge name     bridge id               STP enabled     interfaces
br0             8000.00d0b7169f0d       no              eth0
virbr0          8000.000000000000       yes
[root@opos9 ~]# ip addres show eth0
2: eth0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:d0:b7:16:9f:0d brd ff:ff:ff:ff:ff:ff
    inet6 fe80::2d0:b7ff:fe16:9f0d/64 scope link
       valid_lft forever preferred_lft forever
[root@opos9 ~]# ip addres show br0
5: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    link/ether 00:d0:b7:16:9f:0d brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.108/24 brd 192.168.2.255 scope global br0
    inet6 fe80::2d0:b7ff:fe16:9f0d/64 scope link
       valid_lft forever preferred_lft forever
```

Сетевой мост всегда получает MAC адрес того интерфейса, который подключается к нему первым!

```
[root@opos9 ~]# brctl showmacs br0
port no mac addr                is local?       ageing timer
  1     00:02:44:3d:d8:86       no                88.73
  1     00:0e:08:d3:f4:3f       no                41.90
  1     00:0e:2e:26:55:8c       no                38.85
  1     00:0e:2e:a4:1e:d0       no                14.26
  1     00:11:11:87:00:00       no                76.67
  1     00:11:2f:7a:c9:fc       no                52.65
  1     00:12:cf:a2:d5:42       no                22.84
  1     00:13:d4:9c:73:f7       no                10.54
  1     00:14:2a:a0:a2:22       no               113.74
  1     00:16:17:b1:c9:98       no               241.52
  1     00:1d:60:9b:44:10       no               151.03
  1     00:1d:60:9b:44:24       no               204.82
  1     00:1d:7d:3d:99:a6       no                88.73
  1     00:1d:7d:3e:0a:d5       no               184.83
  1     00:1e:14:b6:61:88       no                 1.58
  1     00:1f:d0:00:59:4f       no                32.44
  1     00:1f:d0:02:7d:f7       no                44.76
  1     00:1f:d0:63:02:99       no               117.07
  1     00:21:28:3a:45:dc       no               159.44
```

```
  1     00:21:28:5e:23:d4       no              185.11
  1     00:21:85:31:3f:5f       no              157.72
  1     00:23:54:51:ba:94       no               12.94
  1     00:30:4f:14:5a:19       no                0.37
  1     00:d0:b7:16:9f:0d       yes               0.00
  1     00:e0:4d:25:cd:4c       no              137.76
  1     60:50:40:30:20:10       no                3.65
```

- информация о MAC адресах источников трафика, прошедшего через коммутатор (и самого коммутатора тоже). Время жизни (aging time) - это количество секунд, которое MAC-адрес будет находится в таблице forwarding database после получения пакета с этим адресом. Записи в таблице периодически удаляются по тайм-ауту.

Мы подключили к мосту сетевой интерфейс и восстановили сетевую работу хоста. Но к мосту может быть подсоединено **сколь угодно много** сетевых интерфейсов! Вторым интерфейсом мы подсоединим тунельный интерфейс `tap0`, который мы создадим так (без помощи QEMU), как это обсуждалось ранее:

```
[root@opos9 ~]# tunctl -u olej -t tap0
Set 'tap0' persistent and owned by uid 501
[root@opos9 ~]# ip addres show tap0
6: tap0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 500
    link/ether ea:32:a6:c7:75:04 brd ff:ff:ff:ff:ff:ff
[root@opos9 ~]# ifconfig tap0 0.0.0.0 promisc
[root@opos9 ~]# ifconfig tap0 up
[root@opos9 ~]# brctl addif br0 tap0
[root@opos9 ~]# ip addres show tap0
6: tap0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 500
    link/ether ea:32:a6:c7:75:04 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::e832:a6ff:fec7:7504/64 scope link
       valid_lft forever preferred_lft forever
```

Ряд страховочных действий (которые могут быть совсем не обязательны на вашем компьютере):

```
[root@opos9 ~]# echo 1024 > /proc/sys/dev/rtc/max-user-freq
[root@opos9 ~]# chmod a+rw /dev/net/tun
```

И запуск QEMU с виртуальным MINIX3, ранее сконфигурированным на сетевой интерфейс 192.168.2.202 (см. далее), без необходимости прав `root`:

```
[olej@opos9 qemu]$ qemu -m 70M -hda minix3-r6084-800.2.202 -boot c -localtime
-net nic,vlan=0 -net \ tap,vlan=0,ifname=tap0,script=no
Could not open '/dev/kqemu' - QEMU acceleration layer not activated
...
```

После старта виртуально MINIX3...

```
[olej@opos9 etc]$ ping 192.168.2.202
PING 192.168.2.202 (192.168.2.202) 56(84) bytes of data.
64 bytes from 192.168.2.202: icmp_seq=1 ttl=96 time=8.87 ms
64 bytes from 192.168.2.202: icmp_seq=2 ttl=96 time=3.07 ms
...
```

- он работает в той же сети (`192.168.2.202/24`), что и вся физическая LAN! Теперь зайдём удалённо на вновь подключенный MINIX3 хост и проделаем краткое его изучение:

```
[olej@opos9 etc]$ telnet 192.168.2.202
```

```
...
Minix  Release 3 Version 1.6  (ttyp0)
...
login: root
Password:
...
Terminal type? (network) minix
# arp -a
192.168.2.1 is at 0:f:34:61:aa:c0
ntc-server.altron.lan (192.168.2.2) is at 0:11:11:87:0:0
opos9.altron.lan (192.168.2.108) is at 0:d0:b7:16:9f:d
# ifconfig -av
/dev/ip0: address 192.168.2.202 netmask 255.255.255.0 mtu 1500
# pr_routes
ent #   if               dest         gateway dist  pref  mtu flags
    0  ip0     192.168.2.1/32   192.168.2.202    1     0    0 static
    1  ip0         0.0.0.0/0      192.168.2.1    1     0    0 static
```

А вот как этот хост был сконфигурирован для такой работы с сетью:

```
# cat /etc/rc.net
ifconfig -I /dev/ip0 -n 255.255.255.0 -h 192.168.2.202
add_route -g 192.168.2.202 -d 192.168.2.1 -n 255.255.255.255 -I /dev/ip0
add_route -g 192.168.2.1 -d 0.0.0.0 -n 0.0.0.0
daemonize nonamed -L
/usr/bin/tcpd telnet /usr/bin/in.telnetd &
/usr/bin/tcpd ftp /usr/bin/in.ftpd &
/usr/bin/tcpd login /usr/bin/in.rlogind &
```

И последнее относительно запуска QEMU: если вы не хотите создавать интерфейс `tap0` вручную, и согласны выполнять виртуальную машину с правами `root`, то вы можете оформить всё, что касается `tap0` в скрипте `/etc/qemu-ifup` (стартовый скрипт, используемый QEMU по умолчанию):

```
[olej@opos9 etc]$ cat /etc/qemu-ifup
#!/bin/sh
echo -------------- tap up -------------
sudo ifconfig $1 0.0.0.0 promisc
sudo brctl addif br0 $1
sudo ifconfig $1 up
```

И запускать QEMU так:

```
[olej@opos9 qemu]# qemu -m 70M -hda minix3-r6084-800.2.202 -boot c -kernel-kqemu \
-localtime -net nic,vlan=0 -net tap,vlan=0
...
```

Всё, что нужно для описываемой выше конфигурации, установится при старте QEMU.

Таким образом можно в **единую сеть** присоединить **сколь угодно много** виртуальных машин с QEMU.

По аналогии с тем, как добавлялись, могут быть и исключены интерфейсы из сетевого моста:

```
[root@opos9 ~]# brctl delif br0 tap0
```

И удалён и сам мост, если необходимость в нём отпала:

```
[root@opos9 ~]# brctl delbr br0
```

Разрешение сетевых имён

Судя по всему, в самом MINIX3 пока не реализованы какие либо средства разрешения сетевых имен (типа `nslookup` — обращения к DNS). Единственный доступный механизм разрешения — через записи, прописанные в файле `/etc/hosts`; такое разрешение имён осуществляет специфический для MINIX3 демон `nonamed`. Пусть у нас, для определённости, в `/etc/hosts` прописано:

```
# cat /etc/hosts
192.168.3.7     %nameserver     #minix
64.102.255.44   %nameserver     #DNS 1
128.107.241.185 %nameserver     #DNS 2
192.168.3.7     minix
72.249.144.181  qnx.org.ru
```

- последняя строка дописана как образец, для включения часто используемых имён в сферу разрешений `nonamed`, и для примеров разрешения на локальном хосте. Имена доступных DNS-серверов (`64.102.255.44` и `128.107.241.185`) прописаны в ходе начального диалога инсталляции системы, но мы их можем произвольно менять редактированием `/etc/hosts`. Прежде всего, мы должны быть уверены в правильности настройки маршрутизации и достижимости указанных хостов DNS:

```
# ping  64.102.255.44
64.102.255.44 is alive
# ping  128.107.241.185
128.107.241.185 is alive
```

Для разрешения имён в MINIX3 используется программа host:

```
# host yandex.ru
yandex.ru has address 213.180.204.11
yandex.ru has address 77.88.21.11
yandex.ru has address 87.250.251.11
yandex.ru has address 93.158.134.11
```

А вот как выглядит её работа с детализацией вывода (`-v`) и конкретизацией адреса используемого DNS (в 1-м примере разрешение осуществляет локальный `nonamed` только из записей в файле `/etc/hosts`):

```
# host -v qnx.org.ru 127.0.0.1
Using domain server 127.0.0.1:
Trying domain ""
rcode = 0 (Success), ancount=1
qnx.org.ru      3600 IN A       72.249.144.181
```

```
# host -v  qnxclub.net 192.168.9.254
Using domain server 192.168.9.254:
Trying domain ""
rcode = 0 (Success), ancount=1
The following answer is not authoritative:
qnxclub.net     86394 IN        A       69.70.20.198
For authoritative answers, see:
qnxclub.net     86394 IN        NS      dns2.is47.com
```

```
qnxclub.net     86394 IN        NS      dns1.is47.com
Additional information:
dns1.is47.com   70100 IN        A       69.70.20.197
dns2.is47.com   70100 IN        A       69.70.20.196
```

Выше показано прямое разрешение (имя в IP). А вот пример обратного разрешения:

```
# host 213.180.204.11
11.204.180.213.in-addr.arpa PTR yandex.ru
```

Дополнительные источники информации

Ссылки на man я даю по размещённым в Интернет файлам для версии MINIX v.2.0.4. Они незначительно отличаются от версии 3; в работающей системе все `man` можно получить непосредственно в консоли системы (или в терминале удалённого доступа TELNET, RLOGIN, SSH) по имени команды.

1. Документация пользователя эмулятора процессора QEMU

Перевод: Павел Марьянов , март 2006

http://jack.kiev.ua/docs/qemu-doc-ru.html

2. «Виртуальный полигон: эмулируем аппаратное обеспечение различных платформ с помощью QEMU»,

Владимир Ляшко

http://www.xakep.ru/magazine/xa/118/094/1.asp

3. Описания не документированных команд MINIX3 `add_route, pr_routes` с примерами использования:

http://www.os-forum.com/minix/net/general-comment-display.php?commentid=171

4. Сайт проекта QEMU

http://www.qemu.org/

и, в частности, полный комплект документации

http://www.qemu.org/user-doc.html

5. `man ip`

http://minix1.woodhull.com/current/2.0.4/wwwman/man4/ip.4.html

- описание на уровне программного кода, но очень проясняющее понятия сетевого интерфейса в MINIX3, и описывающее аварийные коды завершения, которые можно наблюдать при запуске сетевых программ в неправильных конфигурациях.

6. `man` страница по команде `add_route`

http://minix1.woodhull.com/current/2.0.4/wwwman/man8/add_route.8.html

- управление маршрутизацией IP, в этом же `man` находится описания команды `del_route`.

7. `man` страница по команде `pr_routes`

http://minix1.woodhull.com/current/2.0.4/wwwman/man8/pr_routes.8.html

- индикация таблицы маршруизации.

8. «Руководство по `iptables` (Iptables Tutorial 1.1.19)»

http://www.opennet.ru/docs/RUS/iptables/
Автор: Oskar Andreasson
Перевод: Андрей Киселев

9. «Пускаем QEMU в сеть»
http://radist-elvin.blogspot.com/2008/07/qemu_23.html
- настройка `iptables` выхода в сеть из виртуальной сети QEMU.

10. «qemu - работа с сетью»
http://sda00.blogspot.com/2009/05/qemu.html
- несколько QEMU VM в одной сети.

11. «Современный Linux-сервер: виртуализируем сетевые устройства», Алексей Барабанов
http://www.samag.ru/cgi-bin/go.pl?q=articles;n=06.2006;a=01
- использование сетевых мостов.

12. `man` страница по команде `brctl` Linux
http://xgu.ru/wiki/Man:brctl
Перевод: Игорь Чубин

13. «Сетевые мосты в Linux (Linux Bridge)»
http://linuxopen.ru/2008/04/09/setevye-mosty-v-linux-linux-bridge.html
Перевод: Игорь Чубин