Разбираемся с установкой и загрузкой Linux на примере ArchLinux

- Настройка Linux,
- Системное администрирование
- Из песочницы

Сначала мы установим Archlinux и превратим его в загрузочный сервер. Прямо оттуда подготовим новую компактную систему, в которую добавим минимальное графическое окружение и самый необходимый функционал (на примере Firefox). Научим нашу систему загружаться по сети даже на компьютерах с UEFI. Затем полностью переведём её в режим «только для чтения» (сделаем «живой»), что позволит нам использовать систему одновременно хоть на пол сотне разномастных компьютеров с одним единственным загрузочным сервером. Это всё будет работать даже внутри дешёвой 100-Мб сети, которую мы дополнительно «разгоним» в пару раз.

Никакие закладки в жестких дисках будут вам не страшны, потому что дисков у нас не будет. Никакие очумелые ручки пользователей ничего не сломают, т. к. после перезагрузки система вернется в первозданное лично вами состояние. Конечно же, вы научитесь и сможете самостоятельно изменять загружаемую систему таким образом, чтобы в ней содержался только нужный вам функционал и ничего лишнего. Между делом мы выясним, как и в каком порядке загружается Linux, а также из чего он состоит. Знания, как известно, — бесценны, поэтому я делюсь ими даром.

Постараюсь без долгих рассуждений пояснять происходящее, иногда забегая немного вперёд, но впоследствии обязательно раскладывая всё по полочкам. Чтобы у вас вообще не возникало проблем с пониманием, предполагаю, что вы уже работали с каким-нибудь готовым дистрибутивом Linux, пробовали писать простые скрипты с помощью nano или другого текстового редактора. Если вы новичок в ArchLinux, то узнаете много нового, а если «старичок», то узнаете поменьше, но, надеюсь, что в любом случае вы ещё сильнее полюбите Linux.

Информации оказалось очень много. И по устоявшейся голливудской традиции впереди вас ждёт сериал в нескольких частях:
продолжение;
окончание.

Сейчас мы установим Archlinux в VirtualBox, который можно будет клонировать и запускать практически на любом компьютере с legacy BIOS без каких-либо дополнительных настроек. Между делом мы познакомимся с основными приёмами работы с systemd, а также узнаем как его использовать для запуска произвольных служб и программ во время загрузки. Ещё мы увидим, какие этапы проходит Linux при загрузке, и напишем собственный обработчик (hook), который поместим в initramfs. Не знаете что такое initramfs? Тогда заходите под кат.

Есть много причин, по которым выбор пал именно на Archlinux. Первая причина: он мой давний изворотливый приятель и верный помощник. Gentoo, как пишут на просторах Интернета, ещё более изворотлив, но собирать систему из исходников не хочется. Вторая причина: в готовых сборках всегда содержится много лишнего, а перекачивание больших объемов данных может критично сказаться на

производительности сети, да и ничего не видно за широкой спиной «автоматического инсталлятора» — это третья причина. Четвертая: systemd постепенно проникает во все дистрибутивы и даже в Debian, так что мы сможем хорошенько покопаться в грядущем готовых дистрибутивов на примере Archlinux. При всём при этом, систему, которую мы позднее подготовим, можно будет загружать по сети не только сервера, работающего в виртуальной машине, но и с обычного компьютера, например, с Raspberry Pi, и даже с Western Digital My Cloud (проверено), который работает под Debian.

Подготовительные работы

Скачиваем свежий образ по ссылке с официального сайта. В Москве с серверов Яндекса, например, загрузка происходит очень быстро, и если у вас процесс затянулся — просто попробуйте качать в другом месте. Рекомендую запомнить в каком, т. к. эта информация нам ещё пригодится.

В VirtualBox создаем новую виртуальную машину (например, с 1 Гб оперативной памяти и 8 Гб накопителем). В настройках сети необходимо выбрать тип подключения «сетевой мост» и подходящий сетевой адаптер с доступом к сети Интернет. Подключаем скаченный образ к CD-ROM'у. Если не терпится начать работать с железом, то берите флешку и записывайте образ с помощью Win32 Disk Imager (если работаете под Windows), а потом загружайте будущий сервер прямо с неё.

Включаем машину, дожидаемся появления командной строки и устанавливаем пароль, без которого SSH работать не будет:

```
passwd
```

Запускаем сервер SSH командой:

```
systemctl start sshd
```

Остается узнать IP адрес машины, изучив вывод команды:

```
ip addr | grep "scope global"
```

Адрес будет указан сразу после «inet».

Теперь пользователи Windows смогут подключиться к машине с помощью putty, а потом будут копировать отсюда команды и вставлять их и нажатием правой кнопки мыши.

Базовая установка

Дальше максимально коротко опишу стандартную установку Archlinux. Если появятся вопросы, то ответы на них вы, вероятно, найдете в Подробном описании установки для новичков. Wiki просто замечательная, а англоязычная wiki даже актуальная, так что старайтесь пользоваться именно ей.

Подготавливаем носитель с помощью cfdisk (это консольная утилита с простым и понятным интерфейсом). Нам достаточно одного раздела, только не забудьте пометить его как загрузочный:

```
cfdisk /dev/sda
```

Форматируем в ext4 и устанавливаем метку, например HABR:

```
mkfs.ext4 /dev/sda1 -L "HABR"
```

Будущий корневой раздел монтируем в /mnt:

```
export root=/mnt
```

```
mount /dev/sda1 $root
```

Archlinux обычно устанавливается через интернет, поэтому сразу после установки у вас будет самая новая и актуальная версия. Список репозиториев находится в файле /etc/pacman.d/mirrorlist. Постарайтесь вспомнить, откуда скачивали дистрибутив и перенесите эти серверы в самое начало списка — так вы серьезно сэкономите время на следующем шаге. Обычно это серверы, географически расположенные там же, где вы сейчас находитесь.

```
nano /etc/pacman.d/mirrorlist
```

Устанавливаем базовый набор пакетов и набор для разработчиков:

```
pacstrap -i $root base base-devel
```

Теперь воспользуемся командой arch-chroot, которая позволяет временно подменить корневой каталог на любой другой, в котором есть структура корневой файловой системы Linux. При этом программы, которые мы оттуда запустим, не будут знать о том, что снаружи ещё что-то существует. Мы практически окажемся в нашей новой системе с правами администратора:

```
arch-chroot $root
```

Обратите внимание, как поменялось приглашение командной строки.

Выбираем языки, которые планируем использовать. Предлагаю оставить en_US.UTF-8 UTF-8 и ru_RU.UTF-8 UTF-8. В текстовом редакторе нужно просто снять комментарии напротив них:

```
nano /etc/locale.gen
```

Теперь генерируем выбранные локализации:

```
locale-gen
```

Если всё прошло хорошо, то вы увидите примерно такой текст:

```
Generating locales...
  en_US.UTF-8... done
  ru_RU.UTF-8... done
Generation complete.
```

Устанавливаем язык, который будет использоваться по-умолчанию:

```
echo LANG=ru_RU.UTF-8 > /etc/locale.conf
```

А также раскладку и шрифт в консоли:

```
echo -e "KEYMAP=ru\nFONT=cyr-sun16\nFONT_MAP=" > /etc/vconsole.conf
```

Указываем часовой пояс (я использую московское время):

```
ln -s /usr/share/zoneinfo/Europe/Moscow /etc/localtime
```

Придумываем название для нашего будущего сервера:

```
echo "HabraBoot" > /etc/hostname
```

Теперь установим пароль администратора. Делаем мы это в первую очередь из-за того, что SSH не позволит нам подключиться к системе без пароля. Тему неразумности использования системы, незащищенной паролем, здесь мы развивать не будем.

```
passwd
```

Дважды вводим пароль и убеждаемся, что **password updated successfully**.

Добавим нового пользователя с именем *username* (можете выбрать любое), наделим его правами администратора и зададим ему пароль из тех же соображений, а ещё и из-за того, что под root в текущей версии Arch мы не сможем собирать пакеты из AUR (Arch User Repository — это репозиторий от сообщества пользователей Arch Linux с программами, которые не вошли в основной репозиторий):

```
useradd -m username
```

Редактируем файл настроек /etc/sudoers с помощью nano:

```
EDITOR=nano visudo
```

Добавив в него сразу после строки «root ALL=(ALL) ALL» ещё одну строчку:

```
username ALL=(ALL) ALL
```

И задаём пароль для пользователя username:

```
passwd username
```

Теперь нужно установить загрузчик на внутренний накопитель, чтобы система смогла самостоятельно с него загрузиться. В качестве загрузчика предлагаю использовать GRUB, потому что позже он нам снова пригодится. Устанавливаем пакеты с помощью стандартного для Archlinux менеджера пакетов pacman:

```
pacman -S grub
```

Записываем загрузчик в MBR (Master Boot Record) нашего внутреннего накопителя.

```
grub-install --target=i386-pc --force --recheck /dev/sda
```

Если всё прошло нормально, то вы увидите **Installation finished. No error reported**.

Выходим из chroot:

```
exit
```

И замечаем, как поменялось приглашение командной строки.

Мы будем использовать метки дисков, подробное объяснение этого утверждения последует позже.

Снимите комментарий со строки **GRUB_DISABLE_LINUX_UUID=true**, чтобы не использовались UUID накопителей:

```
nano $root/etc/default/grub
```

Генерируем файл конфигурации загрузчика, снова используя arch-chroot. Будет произведён вход, выполнение одной единственной команды, и последует автоматический выход:

```
arch-chroot $root grub-mkconfig --output=/boot/grub/grub.cfg
```

Нам нужно заменить все упоминания **/dev/sda1** на **LABEL=HABR** в файле конфигурации:

```
mv $root/boot/grub/grub.cfg $root/boot/grub/grub.cfg.autoconf &&  cat
```

```
$root/boot/grub/grub.cfg.autoconf | sed
```

```
's/\(root=\)\/dev\/sda1/\1LABEL=HABR/g' > $root/boot/grub/grub.cfg
```

Если поменять в этом же файле строку **set lang=en_US** на **set lang=ru_RU**, то загрузчик будет общаться с нами на великом и могучем.

Генерируем файл fstab с ключом -L, который заставит генератор использовать метки дисков:

```
genfstab -p -L $root > $root/etc/fstab
```

На этом базовая установка ArchLinux закончена. Система будет загружаться самостоятельно и встретит вас приветливым русскоязычным интерфейсом командной строки. Если после этого мы введем команду dhcpcd, то скорее всего даже Интернет заработает. Но мы пока не будем торопиться с перезагрузкой.

Запуск при загрузке с помощью systemd на примере NTP и SSH

Поскольку наша система будет общаться с другими компьютерами, нам потребуется синхронизировать время. Если время на сервере и клиенте будет отличаться, то существует большая вероятность того, что они вообще не смогут

соединиться друг с другом. В свою очередь sudo может начать просить пароль после каждого действия, думая, что таймаут авторизации давно истёк. И кто знает, с чем нам ещё предстоит столкнуться? Перестрахуемся.

Чтобы синхронизировать время с серверами через Интернет по протоколу NTP, нам нужно установить недостающие пакеты. Можно воспользоваться arch-root, но но мы обойдёмся ключами, которые сообщат новое место для установки менеджеру пакетов:

```
pacman --root $root --dbpath $root/var/lib/pacman -S ntp
```

Настроим получение точного времени с российских серверов:

```
mv $root/etc/ntp.conf $root/etc/ntp.conf.old && cat
$root/etc/ntp.conf.old | sed 's/\([0-
9]\).*\(.pool.ntp.org\)/\1.ru\2/g' | tee $root/etc/ntp.conf
```

Нам достаточно синхронизировать время один раз при загрузке. Раньше мы бы записали запуск службы точного времени в файл rc.local, но сейчас появился менеджер системы и служб systemd, который старается запускать службы (в оригинале они называются unit) параллельно для уменьшения времени загрузки системы. Естественно, что работоспособность одной службы может зависеть от функционирования другой. Например, нам бесполезно пытаться синхронизировать время через Интернет до того, как у нас на компьютере заработает сеть. Чтобы описать все эти взаимосвязи, уже недостаточно простого указания имени исполняемого файла, поэтому запуск посредством systemd стал весьма нетривиальным занятием. Для этой цели были созданы специальные файлы с расширением ".service". В них указаны зависимости, имена исполняемых файлов и другие параметры, которые нужно учитывать для успешного запуска. В частности, для управления этапами загрузки в systemd используются цели (target), которые по возлагаемым на них задачам схожи с уровнями запуска (runlevel). Подробности читайте в вики.

К радости новичков, вместе с пакетом ntp поставляется уже готовый ntpdate.service. Все файлы, описывающие запуск служб, находятся в папке $root/usr/lib/systemd/system/, и их можно открыть в любом текстовом редакторе или посмотреть обычным образом. Вот, например, $root/usr/lib/systemd/system/ntpdate.service:

```
[Unit]
Description=One-Shot Network Time Service
After=network.target nss-lookup.target
Before=ntpd.service

[Service]
Type=oneshot
PrivateTmp=true
ExecStart=/usr/bin/ntpd -q -n -g -u ntp:ntp

[Install]
WantedBy=multi-user.target
```

В блоке [Unit] в строке Description указывается краткое описание службы, и при каких условиях она должна быть запущена (в данном случае, после запуска сети, но до перед запуском сервера NTP, который мы вообще не планируем запускать). Запрос точного времени происходит единственный раз во время загрузки, и за это отвечает строка Type=oneshot из блока [Service]. В этом же блоке в строке ExecStart указаны действия, которые необходимо выполнить для запуска сервиса. В блоке [Install] в нашем случае указано, что запуск нашей службы необходим для достижения цели multi-user.target. Рекомендуется использовать такое же содержание блока [Install] для запуска самодельных служб.

В качестве первого практического примера мы немного расширим функциональность ntpdate.service, попросив его дополнительно исправлять время на аппаратных часах. Если после этого, на этом же самом компьютере вы загрузите Windows, то увидите время по Гринвичу, так что не пугайтесь.

Изменение стандартного поведения любой службы systemd производится следующим образом: сначала в папке /etc/systemd/system/ создается новый каталог с полным именем службы и расширением ".d", куда добавляется файл с произвольным именем и расширением ".conf", и уже там производятся нужные модификации. Приступим:

```
mkdir -p $root/etc/systemd/system/ntpdate.service.d && echo -e
'[Service]\nExecStart=/usr/bin/hwclock -w' >
$root/etc/systemd/system/ntpdate.service.d/hwclock.conf
```

Здесь просто говорится о том, что во сразу после запуска службы выполнить команду "/usr/bin/hwclock -w", которая переведёт аппаратные часы.

Добавляем службу ntpdate в автозагрузку (синтаксис стандартен для всех служб):

```
arch-chroot $root systemctl enable ntpdate
Created symlink from /etc/systemd/system/multi-
user.target.wants/ntpdate.service to
/usr/lib/systemd/system/ntpdate.service.
```

Как видите, в каталоге multi-user.target.wants создалась обыкновенная символическая ссылка на файл ntpdate.service, а упоминание о цели multi-user.target мы видели в блоке [Install] этого самого файла. Получается для того, чтобы система достигла цели multi-user.target, должны быть запущены все службы из каталога multi-user.target.wants.

Теперь устанавливаем пакет SSH аналогичным способом (в ArchLinux он называется openssh):

```
pacman --root $root --dbpath $root/var/lib/pacman -S openssh
```

Но на этот раз для автозапуска мы будем использовать сокет, чтобы сервер SSH стартовал только после поступления запроса на подключение, а не висел мёртвым грузом в оперативной памяти:

```
arch-chroot $root systemctl enable sshd.socket
```

Мы не поменяли стандартный 22-й порт и не включили принудительное использование Protocol 2 — пусть это останется на моей совести.

Забегая вперед или знакомимся с обработчиками (hooks)

Чтобы мы могли не глядя подключиться к нашему будущему серверу, нам нужно знать его IP адрес. Будет намного проще, если этот адрес — статический. Обычные способы, о которых говорится в вики, нам не подходят. Проблема в том, что сетевые адаптеры в современном мире именуются согласно своему физическому расположению на материнской плате. Например, имя устройства enp0s3 означает, что это сетевой адаптер ethernet, который расположен на нулевой шине PCI в третьем слоте (подробности здесь). Сделано так для того, чтобы при замене одного адаптера другим, имя устройства в системе не поменялось. Такое поведение нам не желательно, т. к. на разных моделях материнских плат положение сетевой карты может быть разным, и когда мы попытаемся перенести наш загрузочный сервер из VirtualBox на реальное железо, нам скорее всего придётся загружаться с клавиатурой и монитором, чтобы правильно настроить сеть. Нам нужно, чтобы имя сетевого адаптера стало более предсказуемым, например, eth0 (это место зарезервировано смайликом).

Почему будем делать так?

Устанавливаем пакет mkinitcpio-nfs-utils, и у нас появится обработчик (hook) под названием «net»:

```
pacman --root $root --dbpath $root/var/lib/pacman -S mkinitcpio-nfs-
utils
```

По-умолчанию, все файлы обработчика попадают в /usr/lib/initcpio/. Обычно это парные файлы с одинаковым названием, один из которых окажется в подкаталоге install, а другой — в hooks. Сами файлы являются обычными скриптами. Файл из папки hooks обычно попадает внутрь файла initramfs (позже мы о нём всё узнаем) и выполняется при загрузке системы. Второй файл из пары попадает в папку install. Внутри него есть функция build(), в которой находятся сведения о том, какие действия нужно выполнить во время генерации файла initramfs, а также функция help() с описанием того, для чего предназначен данный обработчик. Если запутались, то просто читайте дальше, и всё сказанное в этом абзаце встанет на свои места.

Папка initcpio также присутствует в каталоге /etc, и в ней тоже есть подкаталоги install и hooks. При этом она имеет безусловный приоритет над /usr/lib/initcpio, т. е. если в обеих папках окажутся файлы с одинаковыми названиями, то при

генерации initcpio будут использоваться файлы из /etc/initcpio, а не из /usr/lib/initcpio.

Нам нужно немного поменять функциональность обработчика net, поэтому просто скопируем файлы из /usr/lib/initcpio в /etc/initcpio:

```
cp $root/usr/lib/initcpio/hooks/net $root/etc/initcpio/hooks/ && cp
```

```
$root/usr/lib/initcpio/install/net $root/etc/initcpio/install/
```

Приводим файл hooks/net к следующему виду:

```
cat $root/etc/initcpio/hooks/net
```

```
# vim: set ft=sh:

run_hook() {
    if [ -n "$ip" ]
    then
        ipconfig "ip=${ip}"
    fi
}

# vim: set ft=sh ts=4 sw=4 et:
```

Теперь откроем файл $root/etc/initcpio/install/net и увидим, что в функции help() отлично написано, что из себя должна представлять переменная «ip»:

```
ip=<client-ip>:<server-ip>:<gw-
ip>:<netmask>:<hostname>:<device>:<autoconf>
```

Останется просто установить значение переменной, чтобы задать статический IP адрес и название сетевого устройства, например так «192.168.1.100::192.168.1.1:255.255.255.0::eth0:none» (здесь и далее используйте подходящие для себя настройки сети). В следующем разделе вы узнаете, где именно задаётся значение.

А пока уберём всё лишнее из файла $root/etc/initcpio/install/net. Оставляем загрузку модулей сетевых устройств, программу ipconfig, которую использовали выше, и, естественно, сам скрипт из папки hooks, выполняющий всю основную работу. Получится примерно следующее:

```
cat $root/etc/initcpio/install/net
```

```
#!/bin/bash
```

```
build() {
    add_checked_modules '/drivers/net/'
    add_binary "/usr/lib/initcpio/ipconfig" "/bin/ipconfig"
    add_runscript
}

help() {
    cat <<HELPEOF
This hook loads necessary modules for a network device.
```

```
    Manually configures network and freezes network device name.

HELPEOF

}

# vim: set ft=sh ts=4 sw=4 et:
```

Когда во время загрузки менеджер устройств systemd-udevd попробует переименовать наше сетевое устройство в привычное ему predictable network interface name, например, в enp0s3, то у него ничего не получится. Почему — читайте дальше.

Как происходит загрузка системы

Для простоты рассмотрим обычные BIOS. После включения и инициализации, BIOS начинает по порядку идти по списку загрузочных устройств, пока не найдет загрузчик, которому передаст дальнейшее управление загрузкой.

Как раз такой загрузчик мы записали в MBR нашего накопителя. Мы использовали GRUB, в настройках которого (файл grub.cfg) указали, что корневой раздел находится на диске с меткой HABR. Вот эта строка целиком:

```
linux   /boot/vmlinuz-linux root=LABEL=HABR rw  quiet
```

Здесь упомянут файл vmlinuz-linux, который является ядром системы, а указатель на корневую систему является его параметром. Мы просим искать корневую систему на устройстве с меткой HABR. Здесь также мог бы быть уникальный для каждого накопителя UUID, но в этом случае при переносе системы на другой диск нам несомненно пришлось бы его изменить. Если бы мы указали положение корневой системы привычным для линуксоидов образом: /dev/sda1, то не смогли бы загрузиться с USB накопителя, т. к. это имя USB накопитель бы получил только будучи единственным накопителем в компьютере. Маловероятно, что в компьютере окажется ещё один накопитель с меткой HABR, но не стоит об этом забывать.

Здесь же устанавливается значение глобальной переменной «ip» для нашего обработчика «net» (не забудьте поменять адреса на используемые в вашей сети):

```
linux   /boot/vmlinuz-linux root=LABEL=HABR rw  quiet
```

```
ip=192.168.1.100::192.168.1.1:255.255.255.0::eth0:none
```

В соседней строке есть упоминание файла initramfs, с которым я обещал разобраться:

```
initrd  /boot/initramfs-linux.img
```

Далее при загрузке происходит следующее: загрузчик GRUB получает файлы vmlinuz и initramfs, сообщает им, где искать корневую файловую систему и передаёт им управление дальнейшей загрузкой.

Название initramfs образовано от initial ram file system. Это на самом деле обычная корневая файловая система Linux, упакованная в архив. Она разворачивается в оперативной памяти во время загрузки и предназначена для того, чтобы найти и подготовить корневую файловую систему нашего linux, который мы пытаемся загрузить в итоге. В initramfs есть всё необходимое для этих целей, ведь это настоящий «маленький линукс», который может выполнять многие обычные команды. Его возможности расширяются с помощью обработчиков (hooks), которые помогают сформировать новую корневую файловую систему нашего linux.

После того, как программы из initramfs выполнят свою работу, управление дальнейшей загрузкой передается процессу init подготовленной корневой файловой системы. В качестве процесса init Archlinux использует systemd.

Менеджер устройств systemd-udevd является частью systemd. Он, как и его старший брат, старается обнаруживать и настраивать все устройства в системе параллельно. Он начинает свою работу одним из первых, но уже после того, как наш обработчик net инициализирует сетевую карту ещё на на этапе работы initramfs. Таким образом, systemd-udevd не может переименовать используемое устройство, и имя eth0 сохраняется за сетевой картой в течение всего времени работы.

Готовим initramfs

Для создания файла initramfs используется программа mkinitcpio, которая входит в пакет base, установленный нами в самом начале. Настройки находятся в файле $root/etc/mkinitcpio.conf, а пресеты лежат в папке /etc/mkinitcpio.d. От нас требуется сделать initramfs таким, чтобы он смог найти и подготовить корневую файловую систему, с которой впоследствии начнёт работать systemd. Нам совершенно необязательно учитывать все возможные варианты, достаточно только самого необходимого, чтобы не увеличивать размеры файла initramfs.

Более подробная информация находится
здесь wiki.archlinux.org/index.php/Mkinitcpio

Обязательно убираем обработчик autodetect. Он проверяет устройства установленные в данном конкретном компьютере, и оставляет только необходимые для них модули в initramfs. Нам этого не нужно, поскольку мы изначально рассматриваем возможность дальнейшего переноса системы на другой компьютер, который аппаратно скорее всего будет отличаться от используемой виртуальной машины.

Достаточный для наших целей список обработчиков включая созданный нами net выглядит следующим образом:

```
HOOKS="base udev net block filesystems"
```

вставляем эту строку в файл mkinitcpio.conf, а старую комментируем:

```
nano $root/etc/mkinitcpio.conf
```

На базе стандартного пресета linux создаем свой пресет habr:

```
cp $root/etc/mkinitcpio.d/linux.preset
```

```
$root/etc/mkinitcpio.d/habr.preset
```

И приводим его к такому виду:

```
cat $root/etc/mkinitcpio.d/habr.preset
```

```
ALL_config="/etc/mkinitcpio.conf"
```

```
ALL_kver="/boot/vmlinuz-linux"
```

```
PRESETS=('default' )
```

```
default_image="/boot/initramfs-linux.img"
```

Нам не нужна ветка 'fallback', которая удаляет из обработчиков autodetect, ведь мы его уже сами убрали, и нам не нужно дважды генерировать одинаковый файл initramfs с разными названиями.

Генерируем новый initramfs с помощью пресета habr:

```
arch-chroot $root mkinitcpio -p habr
```

Пишем службу обновления DNS для использования с systemd

Наша сетевая карта получает все настройки для того, чтобы работала сеть и Интернет. Но названия сайтов переводиться в IP адреса не будут, т. к. наша система не знает, какие серверы DNS следует для этого использовать. Напишем собственную службу для этих целей, которую при загрузке будет запускать systemd. А чтобы узнать что-то новое и не заскучать от однообразия, передадим информацию о названии сетевого устройства в качестве параметра, а список DNS серверов сохраним во внешнем файле.

Обновлением информации о DNS серверах занимается resolvconf. Нам идеально подходит синтаксис:

```
resolvconf [-m metric] [-p] -a interface <file
```

В импортируемом здесь файле IP адрес каждого сервера указывается в новой строке после ключевого слова nameserver. Можно указать сколько угодно серверов, но использоваться будут только первые 3 из них. В качестве примера воспользуемся серверами Яндекс. В этом случае файл, передаваемый в resolvconf, должен выглядеть вот так:

```
nameserver 77.88.8.8
```

```
nameserver 77.88.8.1
```

Нам нужно получать информацию о DNS серверах до того, как система будет уверена, что сеть полностью работает, т. е. до достижения цели network.target.

Будем считать, что информацию о серверах нам достаточно обновлять один раз во время загрузки. И стандартно скажем, что нашу службу требует цель multi-user.target. Создаём файл запуска службы в каталоге со следующим содержанием:

```
cat $root/etc/systemd/system/update_dns@.service
```

```
[Unit]
Description=Manual resolvconf update (%i)
Before=network.target

[Service]
Type=oneshot
EnvironmentFile=/etc/default/dns@%i
ExecStart=/usr/bin/sh -c 'echo -e "nameserver ${DNS0}\nnameserver
${DNS1}" | resolvconf -a %i'

[Install]
WantedBy=multi-user.target
```

В строке ExecStart мы выполняем команду echo, на лету генерирующую файл со списком серверов, который через конвейер передаем resolvconf. Вообще, в строке ExecStart нельзя использовать несколько команд и тем более нельзя использовать конвейеры, но мы снова всех обманули, передав эти команды в качестве параметра -c для /usr/bin/sh.

Обратите внимание, что в названии файла update_dns@.service используется символ @, после которого можно указать переменную, и она попадёт внутрь файла, заменив собой "%i". Таким образом строка EnvironmentFile=/etc/default/dns@%i превратится в EnvironmentFile=/etc/default/dns@eth0 — именно это название внешнего файла,

мы будем использовать для хранения значения переменных DNS0 и DNS1. Синтаксис как в обычных скриптах: «название переменной=значение переменной». Создадим файл:

```
nano $root/etc/default/dns@eth0
```

И добавим следующие строки:

```
DNS0=77.88.8.8
```

```
DNS1=77.88.8.1
```

Теперь добавляем службу в автозагрузку не забывая указать имя сетевой карты после @:

```
arch-chroot $root systemctl enable update_dns@eth0.service
```

Только что мы написали универсальный файл, обеспечивающий запуск службы. Универсальность заключается в том что, если в нашей системе окажется несколько сетевых адаптеров, то для каждого из них мы сможем указать свои собственные DNS серверы. Нужно будет просто подготовить набор файлов со списком серверов для каждого из устройств и запускать службу для каждого адаптера в отдельности указывая его имя после @.

Перед первым запуском

На этом первоначальная настройка закончена. Нам нужно загрузить установленный ArchLinux с внутреннего накопителя, чтобы произведённые нами изменения вступили в силу.

Отключаем готовую корневую систему:

```
umount $root
```

И выключаем виртуальную машину:

```
poweroff
```

Теперь можно отключить загрузочный образ из CD-ROM или достать флешку, после этого включаем машину и убеждаемся, что всё работает.

Продолжение и окончание.

Теги:

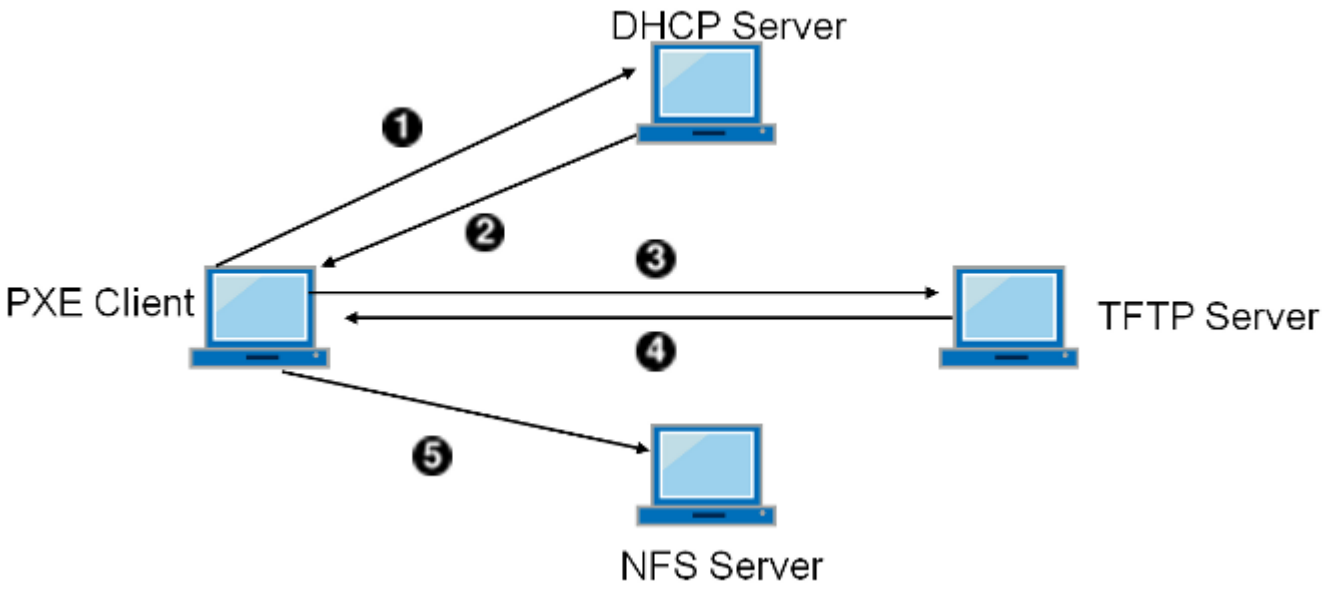
- установка linux
- новичкам
- systemd
- hooks
- virtualbox
- initramfs
- archlinux
- arch linux

Разбираемся с загрузкой ArchLinux по сети

- Настройка Linux,
- Системное администрирование,
- Сетевые технологии
- Tutorial

В предыдущей статье мы подготовили базовую систему. Закончим настройку в следующей статье.

Здесь мы создадим новую систему Arch Linux, способную загружаться по сети и автоматически запускать браузер Firefox, а между делом разберёмся с необходимой функциональностью загрузочного сервера. Потом настроим сам сервер и попробуем с него загрузиться. Всё произойдёт в точности, как на картинке, которую нашёл гугл по запросу «PXE»:



Снова устанавливаем Linux

Archlinux выгодно отличается от готовых дистрибутивов тем, что установка новой системы из рабочей машины осуществляется точно так же, как при использовании установочного образа, и в обоих случаях вы получаете самую актуальную на данный момент версию системы. Понадобятся лишь небольшие установочные скрипты:

```
pacman -S arch-install-scripts
```

Совершенно предсказуемое начало:

```
export root=/srv/nfs/diskless
```

```
mkdir -p $root
```

Установим только базовые пакеты, поэтому:

```
pacstrap -d -c -i $root base
```

Примечание:

Далее повторите все действия вплоть до установки загрузчика согласно предыдущей статье. Вот чек-лист:

- проведите русификацию (интернационализацию);
- укажите часовой пояс и настройте автозапуск службы NTP;
- добавьте пользователя username и заблокируйте его пароль от изменения.

Сравним загрузку с диска и загрузку по сети

В предыдущей статье мы рассматривали процесс загрузки Linux с точки зрения внутреннего накопителя. Сейчас мы представим происходящее глазами сетевой карты. Картинка из заголовка хорошо иллюстрирует события за исключением того, что все серверы в нашем случае будут работать на одном компьютере.

Сразу после включения компьютера, срабатывает код PXE (Preboot eXecution Environment, произносится пикси — спасибо вики), разместившийся непосредственно в ПЗУ сетевой карты. Его задача — найти загрузчик и передать ему управление.

Примечание:

Сетевой адаптер совершенно не представляет в какой сети сейчас находится, поэтому назначает себе адрес 0.0.0.0 и отправляет сообщение DHCPDISCOVER. К сообщению прикреплются ~~паспортные~~данные, которые обязательно нам пригодятся:

- ARCH Option 93 — архитектура PXE клиента (UEFI, BIOS);
- Vendor-Class Option 60 — идентификатор, который у всех PXE клиентов имеет вид «PXEClient:Arch:xxxxx:UNDI:yyyzzz», где цифры xxxxx – архитектура клиента, yyyzzz – мажорная и минорная версии драйвера UNDI (Universal Network Driver Interface).

Адаптер ожидает получить ответ от DHCP сервера по протоколу BOOTP (Bootstrap Protocol), где помимо нужного IP адреса, маски подсети и адреса шлюза, присутствует информация об адресе TFTP-сервера и названии файла загрузчика, который с него следует забрать. Сервер TFTP, в свою очередь, просто отдаёт любому желающему любые файлы, которые у него попросят.

После получения ответа и применения сетевых настроек, дальнейшее управление загрузкой передаётся полученному файлу, размер которого не может превышать 32 кБ, поэтому используется двухстадийная загрузка. Всё необходимое для отображения на экране загрузочного меню докачивается следом по тому же протоколу TFTP. Подавляющее большинство руководств по сетевой загрузке использует загрузчик pxelinux, но GRUB умеет то же самое, и даже больше: в нём есть разные загрузчики для разных архитектур, включая UEFI.

Далее загрузка приостанавливается на время отображения загрузочного меню, а потом по тому же протоколу TFTP докачиваются выбранные файлы vmlinuz и initramfs, которым передается дальнейшее управление загрузкой. На этом этапе уже нет вообще никакой разницы в механизме загрузки по сети или с внутреннего накопителя.

Настраиваем загрузку по сети с помощью GRUB

Поскольку GRUB на нашем сервере уже есть, создадим с его помощью структуру папок для сетевого клиента вот таким образом:

```
grub-mknetdir --net-directory=$root/boot --subdir=grub
```

В папке $root/boot появится папка grub и несколько других. Эту файловую структуру мы будем целиком «отдавать» с помощью TFTP-сервера. Сейчас мы используем 64-битный ArchLinux по той причине, что в 32-битной системе нет папки /grub/x86_64-efi/, которая требуется для загрузки систем UEFI. Можно взять эту папку с нашего 64-битного сервера и в неизменном виде перенести на 32-битный сервер, тогда в нём также появится поддержка UEFI.

Создайте файл конфигурации загрузчика со следующим содержимым:

cat $root/boot/grub/grub.cfg

Я взял файл grub.cfg с сервера и убрал из него всё то, что не участвует в отображении загрузочного меню GRUB или как-то связано с дисками.

Обратите внимание на знакомую нам строку с параметрами ядра:

```
linux /vmlinuz-linux add_efi_memmap
```

```
ip="$net_default_ip":"$net_default_server":192.168.1.1:255.255.255.0:
```

```
:eth0:none nfsroot=${net_default_server}:/diskless
```

Как и в предыдущий раз присваиваем значение переменной «ip». Напоминаю, что она используется в обработчике «net», который мы приспособили для настройки сетевой карты в загрузочном сервере. Здесь снова указывается статический IP адрес и постоянное имя сетевой карты eth0. Значения $net_default_ip и $net_default_server подставляются GRUB самостоятельно на основании данных, полученных из самого первого DHCP запроса. $net_default_ip – это выделенный для нашей машины IP адрес, а $net_default_server — IP адрес загрузочного сервера.

Большинство руководств по сетевой загрузке (среди обнаруженных на просторах рунета), предлагают устанавливать переменную так «ip=::::::eth0:dhcp», что вынуждает обработчик net отправлять новый запрос DHCPDISCOVER для повторного получения сетевых настроек.

Нет объективной причины лишний раз «спамить» DHCP-сервер и ждать, пока он откликнется, поэтому снова используем статику и не забываем указать DNS-серверы. Такую задачу мы уже решали, поэтому просто копируем нужные файлы и добавляем службу в автозагрузку:

```
cp {,$root}/etc/systemd/system/update_dns@.service && cp
{,$root}/etc/default/dns@eth0 && arch-chroot $root systemctl enable
update_dns@eth0
```

Возвращаемся к строке с параметрами ядра. Ещё незнакомая нам команда add_efi_memmap (EFI memory map) добавляет EFI memory map доступной RAM. В прошлый раз мы её намеренно пропустили, из-за сравнительно сложной предварительной разметки носителя для поддержки UEFI. Сейчас нам ничего размечать не нужно, потому что файловая система на загрузочном сервере уже существует и будет использоваться в неизменном виде.

Переменная ядра — nfsroot показывает, где именно в сети нужно искать корневую файловую систему. Она выполняет ту же самую функцию, что и переменная root в загрузочном сервере. В данном случае указан адрес NFS-сервера, который в нашем случае совпадает с TFTP-сервером, но это совершенно необязательно.

Подготавливаем initramfs

За подключение корневой файловой системы по протоколу NFS отвечает обработчик net. В прошлый раз мы убирали из него эту функциональность, но сейчас она нам понадобится, правда, в немного доработанном виде. Дело в том, что обработчик net из коробки поддерживает подключение только по протоколу NFS версии 3. К счастью, поддержка 4-й версии добавляется очень просто.

Сначала установим пакет, в который входит нужный нам обработчик net, а также пакет утилит для работы с NFS (модуль nfsv4 и программа mount.nfs4):

```
pacman --root $root --dbpath $root/var/lib/pacman -S mkinitcpio-nfs-
utils nfs-utils
```

Исправим обработчик net из папки hooks (вместо команды для монтирования nfsmount, теперь будем использовать mount.nfs4):

```
sed s/nfsmount/mount.nfs4/ "$root/usr/lib/initcpio/hooks/net" >
"$root/etc/initcpio/hooks/net_nfs4"
```

С помощью установщика обработчика из папки install добавим модуль nfsv4 и программу mount.nfsv4 в iniramfs. Сначала копируем и переименовываем заготовку:

```
cp $root/usr/lib/initcpio/install/net
$root/etc/initcpio/install/net_nfs4
```

Теперь исправляем только одну функцию build(), а всё остальное не трогаем:

```
nano $root/etc/initcpio/install/net_nfs4
```

```
build() {
    add_checked_modules '/drivers/net/'
    add_module nfsv4?
    add_binary "/usr/lib/initcpio/ipconfig" "/bin/ipconfig"
    add_binary "/usr/bin/mount.nfs4" "/bin/mount.nfs4"
    add_runscript
}
```

Добавляем обработчик в initramfs путём исправления строки в файле mkinitcpio.conf:

```
nano $root/etc/mkinitcpio.conf
```

```
HOOKS="base udev net_nfs4"
```

Если ничего не трогать, то обычно для сжатия файла initramfs используется быстрый архиватор gzip. Мы не настолько торопимся, насколько хотим компрессию посильнее, поэтому воспользуемся xz. Снимаем комментарий с этой строки в файле mkinitcpio.conf:

```
COMPRESSION="xz"
```

Архивация xz происходит значительно дольше, но файл initramfs при этом уменьшается минимум в пару раз, из-за чего гораздо быстрее передается TFTP сервером по сети. Копируем пресет с нашего сервера, чтобы в ходе работы генерировался только один файл initramfs, после чего запускаем mkinitcpio:

```
cp /etc/mkinitcpio.d/habr.preset $root/etc/mkinitcpio.d/habr.preset
```

```
&& arch-chroot $root mkinitcpio -p habr
```

Напоследок отредактируем fstab. Здесь можно подобрать опции монтирования корневой файловой системы, чтобы оптимизировать её работу, но мы ничего трогать не будем:

```
echo "192.168.1.100:/diskless / nfs defaults 0 0" >> $root/etc/fstab
```

Базовая установка клиентской системы на этом закончена. Но мы хотим добавить графическое окружение и автоматический запуск Firefox.

Загружаемся в Firefox

Для уменьшения объема памяти, занимаемого нашей системой, мы откажемся от использования экранного менеджера и остановимся на простейшем оконном менеджере, например, openbox с автоматической авторизацией пользователя username. Использование «облегченных» компонентов позволит системе замечательно запускаться и работать даже на самом древнем железе.

Установим модули для поддержки VirtualBox, сервер X, симпатичный TTF-шрифт, openbox и firefox (все остальные модули будут установлены как зависимости):

```
pacman --root $root --dbpath $root/var/lib/pacman -S virtualbox-
guest-modules virtualbox-guest-utils xorg-xinit ttf-dejavu openbox
firefox
```

Включаем автозагрузку службы virtualbox:

```
arch-chroot $root systemctl enable vboxservice
```

Добавим автоматический вход пользователя username без ввода пароля, для этого изменим строку запуска agetty:

```
mkdir $root/etc/systemd/system/getty@tty1.service.d && \
echo -e "[Service]\nExecStart=\nExecStart=-/usr/bin/agetty --
autologin username --noclear %I  38400 linux Type=simple %I" >
$root/etc/systemd/system/getty@tty1.service.d/autologin.conf
```

Сразу же после авторизации пользователя выполняется файл ~/.bash_profile, из его домашней папки, куда мы добавляем автоматический запуск графического сервера:

```
echo '[[ -z $DISPLAY && $XDG_VTNR -eq 1 ]] && exec startx &>
/dev/null' >> $root/home/username/.bash_profile
```

За запуском X-сервера должен стартовать openbox:

```
cp $root/etc/X11/xinit/xinitrc $root/home/username/.xinitrc && echo
'exec openbox-session' >> $root/home/username/.xinitrc
```

Закомментируйте следующие строки в самом конце файла (от строки twm до добавленной нами строки с запуском openbox, но не включая её):

```
cat $root/home/username/.xinitrc
```

```
# twm &
# xclock -geometry 50x50-1+1 &
# xterm -geometry 80x50+494+51 &
# xterm -geometry 80x20+494-0 &
# exec xterm -geometry 80x66+0+0 -name login
exec openbox-session
```

Копируем конфигурационные файлы openbox

```
mkdir -p $root/home/username/.config/openbox && cp -R
$root/etc/xdg/openbox/* $root/home/username/.config/openbox
```

Добавляем firefox в автозагрузку в окружении openbox:

```
echo -e 'exec firefox habrahabr.ru/post/253573/' >>
$root/home/username/.config/openbox/autostart
```

Поскольку мы только что от имени суперпользователя хозяйничали в домашней папке пользователя username, нам нужно вернуть ему права на все файлы, расположенные в его папке:

```
chown -R username $root/home/username
```

Подготовка системы к загрузке по сети закончена, и настала пора переходить к настройке загрузочного сервера. Теперь мы знаем, что для загрузки нам понадобятся:

- DHCP-сервер с поддержкой протокола BOOTP для настройки сетевой карты;
- TFTP-сервер для передачи загрузчика и файлов vmlinuz и initramfs, которые у нас находятся в папке $root/boot/grub;
- NFS-сервер для размещения корневой файловой системы, которая лежит у нас в папке $root.

Настраиваем загрузочный сервер

Дальнейшие шаги с небольшими изменениями повторяют эту статью из вики, поэтому минимум комментариев с моей стороны.

Устанавливаем DHCP сервер

Скачиваем пакет:

```
pacman -S dhcp
```

и приводим содержимое конфигурационного файла /etc/dhcpd.conf к следующему виду:

```
mv /etc/dhcpd.conf /etc/dhcpd.conf.old
```

nano /etc/dhcpd.conf

Как видите, DHCP-сервер будет отвечать только на те запросы DHCPDISCOVER, которые

придут от PXE клиентов, а остальные просто проигнорируются.

Запускаем DHCP сервер:

```
systemctl start dhcpd4
```

Устанавливаем TFTP сервер

Скачиваем и устанавливаем необходимый пакет:

```
pacman -S tftp-hpa
```

Нам нужно, чтобы TFTP сервер предоставлял доступ к файлам загрузчика, которые мы разместили в папке $root/boot. Для этого модифицируем запуск службы уже проверенным способом:

```
mkdir -p /etc/systemd/system/tftpd.service.d && echo -e
'[Service]\nExecStart=\nExecStart=/usr/bin/in.tftpd -s
/srv/nfs/diskless/boot' >
/etc/systemd/system/tftpd.service.d/directory.conf
```

Первая строка «ExecStart=» отменяет выполнение команды, указанной в оригинальном файле $root/usr/lib/systemd/system/tftpd.service, а вместо нее выполняется "/usr/bin/in.tftpd -s /srv/nfs/diskless/boot". Только в том случае, когда служба запускается однократно (Type=oneshot), мы можем использовать несколько строк ExecStart= чтобы выполнять команды одну за другой. Это не тот случай, поэтому отменяем одну команду и выполняем другую.

Запускаем TFTP сервер:

```
systemctl start tftpd.socket tftpd.service
```

Устанавливаем NFS сервер

Скачиваем пакет:

```
pacman -S nfs-utils
```

Добавляем папку, в которую мы установили систему, в список экспортируемых:

```
echo -e "/srv/nfs
192.168.1.0/24(rw,fsid=root,no_subtree_check,no_root_squash)\n$root
192.168.1.0/24(rw,no_subtree_check,no_root_squash)" >> /etc/exports
```

Не забываем использовать синтаксис NFS v.4 указывая путь относительно папки с fsid=root (корневой по отношению ко всем остальным экспортируемым папкам, без указания которой ничего работать не будет).

Запускаем службы, обеспечивающие работу NFS-сервера:

```
systemctl start rpcbind nfs-server
```

На этом загрузочный сервер готов к работе.

Пробуем загрузиться по сети

Проследим за процессом загрузки с сервера с помощью программы tcpdump

```
pacman -S tcpdump
```

```
tcpdump -v '( \
```

```
    src host 0.0.0.0 and udp[247:4] = 0x63350101) or ( \
    dst host HabraBoot and dst port tftp) or ( \
    dst host HabraBoot and tcp[tcpflags] == tcp-syn)'
```

Первая строка «ловит» запрос DHCPDISCOVER от PXE клиента. В выводе, отфильтрованном второй строкой, будут перечислены имена всех файлов, запрашиваемых по TFTP. Третья строка показывает два tcp-syn запроса, отправляемых в самом начале подключения по протоколу NFS (первое соединение осуществляется обработчиком net, а второе переподключение происходит во время обработки файла fstab).

Создаём новую виртуальную машину, для краткости будем называть её «клиент». В настройках сети снова указываем тип подключения «Сетевой мост» и включаем машину. Сразу же нажимаем клавишу F12 на клавиатуре для выбора загрузочного устройства, а потом клавишу l, чтобы загрузиться по сети.

Дождитесь окончания загрузки. Если всё в порядке, то на сервере добавляем используемые службы в автозагрузку:

```
systemctl enable tftpd.socket tftpd.service dhcpd4 rpcbind nfs-server
```

Все серверы DHCP, TFTP и NFS мы запустили на одном загрузочном сервере. Делать так необязательно. Например, роутеры Mikrotik поддерживают Bootp и позволяют использовать себя в качестве TFTP — просто закачайте туда все нужные файлы и проверьте сетевые настройки.

Сейчас графическое окружение будет работать только в VirtualBox, потому что мы не устанавливали драйверы для «железных» видеокарт. Мы решим проблему автоматического подбора нужных драйверов в следующей статье. Заодно ускорим загрузку системы и сделаем из неё «живой образ».

Теги:

- новичкам
- загрузка ос
- archlinux
- arch linux
- grub
- initramfs
- tftpboot
- nfsv4
- dhcpd

Стань повелителем загрузки Linux

- Настройка Linux,
- Системное администрирование,
- Сетевые технологии
- Tutorial

Сначала мы познакомимся с udev и научимся с его помощью исследовать установленные в компьютере устройства прямо во время загрузки: в качестве примера будем автоматически выбирать настройки видеокарт для Xorg. Затем решим задачу работы с одним образом на десятках компьютеров одновременно путём внедрения собственного обработчика в initramfs, а заодно оптимизируем систему для сетевой загрузки. Чтобы дополнительно уменьшить время загрузки и снизить нагрузку на сеть попробуем NFS заменить на NBD, и помочь TFTP с помощью HTTP. В конце вернёмся в начало — к загрузочному серверу, который также переведём в режим «только для чтения».


802154

Данная статья — скорее исследование, а не готовое руководство (все решения работают, просто они не всегда оптимальны). В конце у вас появится достаточно знаний, чтобы сделать всё так, как захотите именно вы.

Начало смотрите здесь:
Первоначальная настройка сервера
Подготовка образа для загрузки по сети

Мы остановились на том, что загрузили по сети машину VirtualBox и запустили Firefox. Если сейчас попытаться сделать то же самое с настоящим компьютером, то на экране появится циклическая авторизация пользователя username и безуспешные попытки запустить графическое окружение — Xorgне находит нужный драйвер.

Запускаем видеокарты

Для работы графического режима в VirtualBox у нас установлено всё необходимое. Изначально планировалось, что наша бездисковая система будет функционировать на любом «железе», но ~~из-за лени~~ мы не станем пытаться объять необъятное, поэтому ограничимся поддержкой графических решений следующих доминирующих производителей: nVidia, Intel и AMD.

Переключимся на машине-клиенте во второй терминал нажатием Ctrl+Alt+F2 и установим открытые драйверы:

```
pacman -S xf86-video-ati xf86-video-nouveau xf86-video-intel
```

Вероятнее всего, что на этот раз Xorg не сможет самостоятельно выбрать подходящие настройки для каждого случая, и судя по экрану загруженного клиента будет казаться, что вообще ничего не изменилось.

Простейший способ узнать какие видеоустройства имеются в системе, это ввести в консоли команду:

```
lspci | grep -i vga
```

```
00:02.0 VGA compatible controller: InnoTek Systemberatung GmbH
```

```
VirtualBox Graphics Adapter
```

Но мы не будем искать лёгких путей, а в награду получим новую порцию знаний.

Ближе знакомимся с udev

Раньше я уже упоминал, что **менеджер устройств** в Archlinux называется udev. Он входит в пакет systemd под именем systemd-udevd.

По мере обнаружения новых устройств в загружаемой системе, ядро создаёт их иерархию в каталоге /devices. Сначала появляется сама система PCI, затем в ней обнаруживаются шины, на которых «сидят» конечные устройства, а их драйверы рассортировывают устройства по классам. Устройства внутри классов обнаруживаются параллельно, точно как systemd параллельно запускает службы для достижения следующей цели.

Асинхронный поиск устройств приводит к тому, что если в компьютере одновременно присутствует несколько устройств, относящихся к одному классу, то они могут обнаруживаться в разном порядке от включения к включению, например, сначала одна видеокарта, а потом — другая, и их имена при этом будут меняться между собой. К счастью, появление нового элемента в иерархии устройств является событием udev, которое можно отследить и принять необходимые меры.

Для менеджера udev придуманы правила, призванные упорядочить хаос, и упростить жизнь установленных программ. Правила хранятся в папках /usr/lib/udev/rules.d/ и /etc/udev/rules.d/ (последняя, как и в случае обработчиков (hooks), имеет более высокий приоритет и файлы оттуда проверяются первыми). Появление нового элемента в иерархии устройств сопровождается проверкой всех установленных правил udev, и автоматическим выполнением указанных там действий, в случае совпадения. Обычно эти действия заключаются в переименовании устройств и установки на них ссылок в каталогах внутри /dev и /sys для удобства использования в программах.

Драйверы видеокарт относят их к подсистеме (классу) drm, поэтому сведения о подобных устройствах дублируются в каталоге /sys/class/drm. Первая видеокарта, обнаруженная в системе, по-умолчанию получает имя «card0», если в ней имеется несколько видеовыходов, то они получают имена вида «card0-CON-n», где «CON» — тип разъема (VGA, HDMI, DVI и др.), а «n» — порядковый номер разъема (причём одни производители нумеруют разъёмы начиная с «0», а другие — с «1»). Следующая видеокарта становится “card1” и т. д.

Если ничего не предпринять, то в виду параллельности обнаружения при следующем включении card1 может стать card0 и наоборот. Udev станет добавлять такие устройства в /dev то с одним, то с другим именем. Случаи, когда такое поведение udev нежелательно, подробно описаны в Интернете, и в них в основном обсуждаются различные USB устройства. Нам же требуется при обнаружении видеокарт запускать определённую программу, которую напишем чуть позже, а пока выясним, что известно udev.

Чтобы узнать то же самое, что знает про видеокарту udev, введём команду на клиенте:

```
udevadm info -a -p /sys/class/drm/card0
```

вывод команды

Обратите внимание на древовидную структуру с использованием парадигмы родительских и дочерних устройств. В строках, начинающихся с «looking at ...» указан путь к данному устройству относительно каталога /sys, т. е. обратившись к видеокарте по пути /sys/class/drm/card0, мы обнаружили, что на самом деле это ссылка на /sys/devices/pci0000:00/0000:00:02.0/drm/card0.

У родительского устройства /devices/pci0000:00/0000:00:02.0 есть атрибут vendor с идентификатором производителя. Udev располагает доступом к обширной базе данных и может перевести этот код в удобоваримый вид:

```
udevadm info -q property -p /sys/devices/pci0000:00/0000:00:02.0
```

```
DEVPATH=/devices/pci0000:00/0000:00:02.0
ID_MODEL_FROM_DATABASE=VirtualBox Graphics Adapter
ID_PCI_CLASS_FROM_DATABASE=Display controller
ID_PCI_INTERFACE_FROM_DATABASE=VGA controller
ID_PCI_SUBCLASS_FROM_DATABASE=VGA compatible controller
ID_VENDOR_FROM_DATABASE=InnoTek Systemberatung GmbH
MODALIAS=pci:v000080EEd0000BEEFsv00000000sd00000000bc03sc00i00
PCI_CLASS=30000
PCI_ID=80EE:BEEF
PCI_SLOT_NAME=0000:00:02.0
PCI_SUBSYS_ID=0000:0000
SUBSYSTEM=pci
```

```
USEC_INITIALIZED=24450
```

Сравните с выводом команды:

```
lspci | grep -i vga
```

```
00:02.0 VGA compatible controller: InnoTek Systemberatung GmbH
```

```
VirtualBox Graphics Adapter.
```

Динамическая настройка видеокарты с помощью udev

Подключитесь к загрузочному серверу. И создайте файл с правилами:

```
export root=/srv/nfs/diskless
```

```
nano $root/etc/udev/rules.d/10-graphics.rules
```

```
KERNEL=="card[0-9]*", SUBSYSTEM=="drm", RUN+="/etc/default/xdevice
```

```
%n"
```

```
KERNEL=="card*", SUBSYSTEM=="drm", ATTR{enabled}=="enabled",
```

```
ATTR{status}=="connected", RUN+="/etc/default/xdevice %n %k"
```

Каждое правило записывается в новой строке. Первая часть служит для идентификации события udev, к которому нужно применить действие, указанное в конце строки. Для опознания события используются данные, которые можно получить в выводе команды «udevadm info -a -p /sys...».

Правило из первой строки срабатывает для всех устройств с именем (ядром) card0, card1… подсистемы drm. Второе правило сработает только для активных устройств из подсистемы drm, к которым в данный момент подключен монитор (оно не сработает для card0, card1, а только для имен вида card0-HDMI-1, т. к. только у таких устройств есть атрибуты enabled и status). При совпадении события с его описанием выполняется одна и

та же программа, в которую в первом случае передаётся один параметр %n (порядковый номер ядра, который для card0 будет «0»), а во втором — дополнительный параметр %k (само имя ядра «card0»).

Программа /etc/default/xdevice будет изменять содержимое файла в папке /etc/X11/xorg.conf.d/, в котором содержится информация о настройках видеоадаптера для xorg. Для разных производителей мы подготовим разные шаблоны, учитывающие особенности реализации. Достаточно указать минимально необходимую информацию для однозначной идентификации устройства, а остальное xorg сделает сам:

```
Section "Device"
    Identifier "уникальный идентификатор устройства"
    Driver     "используемый драйвер"
    Option     "AccelMethod" "метод ускорения"
    BusID      "PCI:идентификатор шины PCI, куда физически установлен
адаптер"
EndSection
```

Необходимые данные мы укажем в самом шаблоне или получим исследуя вывод команды «udevadm info».

Программа будет срабатывать для каждого выхода каждой видеокарты, к которому подключен монитор. Для упрощения задачи заставим работать последний обнаруженный вариант, чтобы на мультимониторных системах работал хотя бы один монитор. Это не самый оптимальный способ настройки, и было бы лучше проверить графическую подсистему один раз перед достижением graphical.target, но наш вариантрабочий и подходит для изучения правил udev в действии.
Создаём файл программы со следующим содержанием:

```
nano $root/etc/default/xdevice
```

Скрытый текст

Сделаем файл исполняемым:

```
chmod +x $root/etc/default/xdevice
```

Отключаем автоматическую загрузку службы VirtualBox, т. к. теперь она будет запускаться нашей программой только при необходимости:

```
systemctl disable vboxservice
```

Добавляем шаблоны конфигурационных файлов xorg, с оптимизированными под основных производителей настройками:

```
nano $root/etc/X11/xorg-device-intel.conf
```

```
Section "Device"
    Identifier "Intel %ID%"
    Driver     "intel"
    Option     "AccelMethod" "uxa"
    BusID      "PCI:%BUS%"
EndSection
```

AMD, nVidia, VirtualBox

В завершение настройки xorg сделаем ”Windows like” переключение раскладки клавиатуры комбинацией Alt+Shift:

```
nano $root/etc/X11/xorg.conf.d/50-keyboard.conf
```

```
Section "InputClass"
```

```
        Identifier "keyboard-layout"
        MatchIsKeyboard "on"
        Option "XkbLayout" "us,ru"
        Option "XkbVariant" ",winkeys"
        Option "XkbOptions" "grp:alt_shift_toggle"
EndSection
```

Оптимизируем систему

Логи работы всех составляющих Archlinux сохраняются в журнале. Если всё оставить как есть, то журнал может довольно сильно раздуть, поэтому ограничим его размер, скажем 30Мб (добавьте или раскомментируйте строку):

```
nano $root/etc/systemd/journald.conf
...
SystemMaxUse=30M
...
```

Каждое действие протоколируется в папку /var/log/journal. В нашем случае передача данных осуществляется по сети, которая на практике имеет невысокую пропускную способность. Можно удалить папку с журналом, то он будет сохраняться только в оперативной памяти, что идеально подходит для бездискового клиента:

```
rm -r $root/var/log/journal
```

При различных ошибках в работе приложений в папке /var/lib/systemd/coredump

создаются автоматические **дампы ядра**. Мы их отключим по той же причине:

```
nano $root/etc/systemd/coredump.conf
```

```
...
```

```
Storage=none
```

```
...
```

Отключаем **SWAP**:

```
echo -e 'vm.swappiness=0\nvm.vfs_cache_pressure=50' >
```

```
$root/etc/sysctl.d/99-sysctl.conf
```

Удалим ненужные **локализации**. Это простое действие поможет сэкономить более 65 Мб. Заодно посмотрим, как устанавливаются программы из AUR (фактически они собираются из исходников). Зайдите на загрузочный сервер с правами обычного пользователя и выполните следующие действия:

```
curl -o localepurge.tar.gz
```

```
https://aur.archlinux.org/packages/lo/localepurge/localepurge.tar.gz
```

```
tar -xvvzf localepurge.tar.gz
```

```
cd localepurge
```

```
makepkg -s
```

Пакет готов. Устанавливаем его из файла, а не из репозитория, поэтому ключ S заменяется на U (исправьте название файла, если версия собранной вами програмы не

совпадает с моей):

```
sudo pacman --root $root --dbpath $root/var/lib/pacman -U
```

```
localepurge-0.7.3.4-1-any.pkg.tar.xz
```

Теперь настроим. Закомментируйте строку «NEEDCONFIGFIRST» в начале файла и укажите используемые локализации в самом конце:

```
nano $root/etc/locale.nopurge
```

```
...
```

```
# NEEDSCONFIGFIRST
```

```
...
```

```
ru
```

```
ru_RU
```

```
ru_RU.UTF-8
```

```
en
```

```
en_US
```

```
en_US.UTF-8
```

Конфигурируем и запускаем программу:

```
arch-chroot $root /usr/bin/localepurge-config
```

```
arch-chroot $root localepurge
```

Переходим в read-only

Если мы попробуем загрузить существующую систему на нескольких компьютерах одновременно, то все копии будут изменять одни и те же папки на сервере. Если один клиент удалит какой-то файл, то он неожиданно исчезнет и у другого. Самый надежный способ защититься от изменений — перейти в режим только для чтения.

Проблема в том, что для нормальной работы системы необходима возможность записывать данные в некоторые папки. Решение на поверхности — подключить эти папки через fstab как tmpfs — замечательно подойдёт для /var/log, например. Но как поступить, например, с каталогом /etc, ведь наше правило udev меняет там файлы, да и другие программы активно с ним работают? Можно где-то сохранить информацию перед монтированием, а потом переписать обратно. Или сразу перенести всё куда-то ещё, а потом вернуть. Ясно одно: придётся долго тестировать и следить за работой системы, чтобы понять какие ещё папки сделать доступными для записи, или же настроить все программы так, чтобы они оставляли продукты своей жизнедеятельности строго в отведённом месте. Слишком мудрёно. Предлагаю всю систему развернуть в RAM. Останется только предварительно переписать в неё всё самое нужное для работы.

Существует одна папка, в которую во время работы ничего не записывается, если мы ничего не устанавливаем — это /usr. Если подмонтировать её на позднем этапе работы initramfs с доступом только для чтения, то на работу Firefox это никак не повлияет. Обязательно сравните размер каталога /usr с размером всего остального, и получится, что копировать останется не так много, а если при этом исключить всё лишнее… Вы тоже подумали о rsync?

Переделываем файловую систему на лету

Устанавливаем rsync на клиента:

```
pacman -S rsync
```

Заниматься копированием предстоит на этапе работы intramfs, следовательно, понадобится новый обработчик, назовём его «live». Сначала сохраним все необходимые параметры монтирования оригинального корневого каталога, путём анализа файла /etc/fstab с помощью утилиты findmnt. Затем корневой каталог отмонтируем от папки /new_root, где он всегда находится внутри initramfs. На его месте создадим ramfs с возможностью записи и подготовим точку монтирования /srv/new_root, куда вернём оригинальный корневой каталог. Останется только переписать все самые нужные файлы и каталоги, за исключением папки /usr, которую забиндим в режиме только для чтения. Копии файлов в ramfs будут доступны для чтения и для записи.

```
nano $root/etc/initcpio/hooks/live
```

cat $root/etc/initcpio/hooks/live

К файлу /etc/fstab мы обращаемся дважды: первый раз получаем информацию по параметрам монтирования корневого каталога, а второй раз проверяем, есть ли в fstab какая-нибудь информация по /usr. Для позднего монтирования /usr в Archlinux есть специальный обработчик usr, которому мы не будем мешать выполнять свою работу. Если /usr монтируется каким-то особым образом, то наш обработчик его пропускает.

В тексте упомянут файл /etc/default/live_filter с правилами фильтрации, предназначенными для rsync, нам нужно не забыть его подготовить. Сделаем это автоматически из установщика обработчика:

```
nano $root/etc/initcpio/install/live
```

```
#!/usr/bin/bash

build() {
    make_filter > /etc/default/live_filter
    add_binary "/usr/bin/rsync" "/bin/rsync"
    add_binary findmnt
    add_runscript
}

make_filter() {
    cat <<EOF
+ /etc/*
```

```
+ /home/*
+ /home/*/.config
- /home/*/*/
+ /var/*
- /var/cache/*/*
- /var/lib/pacman/*/*
- /var/lib/systemd/*/*
+ /var/log/*/
- /var/log/*
- /var/tmp/*
- /*/*
EOF
}
```

Rsync «не видит» дальше одной директории. Файлы и папки в директории проверяются каждым правилом по порядку до первого совпадения ("+" — объект копируется, "-" — объект не копируется). Если совпадений нет, то файл копируется, а директория создаётся пустой. Далее rsync заходит в «выжившую» директорию и снова применяет правила к её содержимому. Так повторяется до тех пор пока совсем ничего не останется.

В нашем случае корневой каталог не попадает ни под одно правило, поэтому его структура полностью переносится (копируются все файлы и создаются пустые каталоги). Каталоги /boot, /dev, /lost+found, /mnt, /opt, /proc, /root, /run, /srv, /sys, /tmp попадают под действие последнего правила "- /*/*", т.е. никакое их содержимое никуда не копируется, но сами они создаются. Каталог /etc сразу же попадает под правило "+ /etc/*", и всё его содержимое копируется, но сначала только в пределах одного каталога (в дальнейшем вся его структура будет перенесена по порядку, потому что для уровней вложенности /etc/*/ и далее никаких правил нет). Похожее начало ждёт каталог /home — папки всех

пользователей попадают под правило "+ /home/*" и будут воссозданы в копии (пока пустыми). Следующее правило "+ /home/*/.config" копирует каталоги .config, вложенные в домашние папки каждого из пользователей, а "- /home/*/*/" исключает все остальные каталоги (правило идёт после «спасательного», поэтому для /home/*/.config не срабатывает). Про сами файлы из домашнего каталога ничего не говорится, поэтому они полностью переносятся. Файлы из исключённых вложенных каталогов не копируются, потому что эти каталоги не были созданы. Правило "- /var/cache/*/*" сохраняет всю структуру каталогов в /var/cache, но их содержимое не переносится. Остальные правила действуют аналогичным образом.

Замечания

Добавляем обработчик в initramfs:

```
cat $root/etc/mkinitcpio.conf
```

```
...
```

```
HOOKS="base udev net_nfs4 live"
```

Генерируем initramfs:

```
arch-chroot $root mkinitcpio -p habr
```

nfs4+live

Разгоняем сеть

Физически, естественно, разгон сети сейчас невозможен без замены оборудования, зато программные оптимизации не запрещаются. Нам нужно передавать содержимое связанной папки /usr по сети. Не отправлять эти данные мы не можем, зато способны уменьшить объём занимаемого ими места — заархивировать. На сервере сжимаем, а на клиенте — распаковываем, и через ту же самую сеть теоретически передаётся больше данных за единицу времени.

Файловая система squashfs совмещает в себе возможности архиватора и монтирования архивов через fstab, как обычную файловую систему. Основной недостаток данной файловой системы — невозможность работать в режиме записи (только для чтения) — для нас недостатком не является:

```
pacman -S squashfs-tools && mksquashfs $root/usr
$root/srv/source_usr.sfs -b 4096 -comp xz
```

Монтировать будем так:

```
nano $root/etc/fstab
# <file system>                 <dir> <type>   <options>
<dump> <pass>
192.168.1.100:/diskless         /     nfs4     defaults,noatime    0
0
/srv/new_root/srv/source_usr.sfs /usr  squashfs loop,compress=xz    0
0
```

На позднем этапе работы initramfs монтированием папки /usr занимается обработчик usr, который нужно немного подправить:

```
cp $root/{usr/lib,etc}/initcpio/install/usr && cp
$root/{usr/lib,etc}/initcpio/hooks/usr
```

Нужно, чтобы строка монтирования выглядела так:

```
nano $root/etc/initcpio/hooks/usr
```

```
mount "/new_root$usr_source" /new_root/usr -o "$mountopts"
```

Чем не устраивает usr?

```
cat $root/etc/mkinitcpio.conf
```

```
HOOKS="base udev net_nfs4 live usr"
```

```
arch-chroot $root mkinitcpio -p habr
```

nfs4+live+squashed /usr

Данных пришлось передать примерно на 20% меньше, чем в предыдущий раз. Можно упаковать весь корневой каталог в один файл, тогда обработчик live для заполнения ramfs будет забирать с сервера данные в сжатом виде.

Можно скопировать файл /srv/source_usr.sfs в ramfs поменяв правила в фильтре rsync, а потом примонтировать его через fstab из нового места, и, когда вся система целиком окажется в RAM, попробовать отключиться от загрузочного сервера.

Убираем лишнее

Если вы заглядывали сюда, то у вас не возникнет вопрос: «Как мы будем отдавать с сервера файл?». Можно, конечно, передавать данные squashfs посредством NFS (что и происходило выше), но существует менее документированное решение Network Block Device, с которым можно работать как с обычным диском. Поскольку это «блочное устройство», а не «файловая система», мы можем использовать на нём любую файловую систему с возможностью сжатия данных. Для доступа на чтение и запись подойдёт btrfs с архивацией zlib, но нам не нужна запись и squashfs вполне устраивает.

Чтобы из initramfs можно было подключиться к NBD-серверу при загрузке понадобится скачать из AUR пакет mkinitcpio-nbd (нужно скачивать и собирать с правами обычного пользователя):

```
curl -o mkinitcpio-nbd.tar.gz
https://aur.archlinux.org/packages/mk/mkinitcpio-nbd/mkinitcpio-
nbd.tar.gz
tar -xvvzf mkinitcpio-nbd.tar.gz
cd mkinitcpio-nbd
makepkg -s
sudo pacman --root $root --dbpath $root/var/lib/pacman -U mkinitcpio-
nbd-0.4.2-1-any.pkg.tar.xz
```

Добавляем в конец файла $root/boot/grub/grub.cfg новый пункт меню:

```
cat $root/boot/grub/grub.cfg

menuentry "NBD" {
    load_video
    set gfxpayload=keep
    insmod gzio
    echo "Загружается ядро..."
    linux vmlinuz-linux \
         add_efi_memmap \
```

```
ip="$net_default_ip":"$net_default_server":192.168.1.1:255.255.255.0:
:eth0:none \
        nbd_host="$net_default_server" nbd_name=habrahabr
root=/dev/nbd0
    echo "Загружается виртуальный диск..."
    initrd initramfs-linux.img
}
```

Как видите, поменялась только одна строчка:

```
nbd_host="$net_default_server" nbd_name=habrahabr root=/dev/nbd0
```

После подключения к NBD серверу в клиенте появляется блочное устройство с именем /dev/nbd0, поэтому поступаем с ним как с обычным диском:

```
nano $root/etc/fstab
```

```
# <file system>             <dir>  <type>    <options>
<dump> <pass>
/dev/nbd0                   /      squashfs  ro,loop,compress=xz 0
0
```

В последних версиях NBD сервера появилась непрятная особенность (скорее всего это

баг). Когда клиент NBD устанавливает соединение с сервером, а потом внезапно выключается не завершая соединение корректно, и оно продолжает «болтаться» на сервере в виде незавершенного процесса. Если клиент во время загрузки попробует подключиться к NBD заново, то есть вероятность, что сервер не станет создавать новое соединенение считая старое активным. Предлагаю непосредственно перед подключением к NBD отправлять свой IP адрес через netcat на сервер, чтобы тот закрыл старые подключения, связанные с этим IP адресом:

```
cp $root/{usr/lib,etc}/initcpio/install/nbd
```

```
cp $root/{usr/lib,etc}/initcpio/hooks/nbd
```

Нужно отредактировать только один файл. Вставьте между строками следующий фрагмент:

```
nano $root/etc/initcpio/hooks/nbd
```

```
                modprobe nbd # вставляете после этой строки
```

```
                msg "closing old connections..."
```

```
                echo ${ip} | nc ${nbd_host} 45678
```

```
                local ready=$(nc -l -p 45678)
```

```
                [ "$ready" -ne 1 ] && reboot
```

```
                msg "connecting..." # и перед этой строкой
```

В initramfs сетью по-прежнему заведует наш модифицированный net_nfs4, после которого вставляем nbd:

```
nano $root/etc/mkinitcpio.conf
```

```
MODULES="loop squashfs"
```

```
HOOKS="base udev net_nfs4 keyboard nbd live"
```

Генерируем initramfs:

```
arch-chroot $root mkinitcpio -p habr
```

Перед выполнением следующей команды удалите или переместите файл $root/srv/source_usr.sfs за пределы $root — не имеет смысла помещать архив /usr внутрь архива, содержащего оригинал /usr:

```
mksquashfs $root/* /srv/new_root.sfs -b 4096 -comp xz
```

Переходим к настройке сервера

Устанавливаем пакет:

```
pacman -S nbd
```

Настраиваем NBD сервер:

```
mv /etc/nbd-server/{config,config.old} && nano /etc/nbd-server/config
```

```
[generic]
```

```
        user = nbd
```

```
        group = nbd
```

```
[habrahabr]
```

```
        exportname = /srv/new_root.sfs
        timeout = 30
        readonly = true
        multifile = false
        copyonwrite = false
```

Всё достаточно просто. Мы создаём шару с именем habrahabr, ссылаемся на наш файл, устанавливаем таймаут соединения, раздаём в режиме «только для чтения», отдаём только один файл и функция copyonwrite нам не нужна. Copyonwrite позволяет использовать одну и ту же раздачу несколькими клиентами одновременно, при этом каждому клиенту создаётся отдельный файл, куда будут записываться все произведённые им изменения оригинального файла. После отключения клиента файлы с изменениями удаляются автоматически. Использование этой функции замедляет сервер. Информации по NBD в интернете не так много, но man'ы решают.

Проверять и завершать процессы, связанные с незакрытыми соединениями будет вот этот файл:

```
nano /etc/default/close_passive_NBD_connections.sh
```

```
#!/bin/sh
# завершает все процессы с полученными PID
_kill(){
    local PID
    for PID in $*
        do kill $PID
    done
```

```
}
main(){
    local rIP PIDs
    # нам передают с клиента значение переменной ip из параметров
ядра в grub.cfg
    rIP=$(netcat -l -p 45678 | cut -d: -f1)
    # фильтруем пакеты с полученного IP адреса и узнаём их PID
    PIDs=$(netstat -np | grep $rIP | awk '/^tcp.*nbd-
server/{split($NF,a,"/");print a[1]}')
    _kill $PIDs && echo "1" | netcat -z $rIP 45678
}
# повторяем в бесконечном цикле
while [ 0 ]
do main
done
```

Файл делаем исполняемым:

```
chmod +x /etc/default/close_passive_NBD_connections.sh
```

Устанавливаем пакеты, в которых находятся утилиты netcat и netstat:

```
pacman -S gnu-netcat net-tools
```

Модифицируем запуск службы NBD:

```
mkdir -p /etc/systemd/system/nbd.service.d && nano
```

```
/etc/systemd/system/nbd.service.d/close_passive.conf
```

```
[Service]
```

```
Type=oneshot
```

```
ExecStart=/etc/default/close_passive_NBD_connections.sh
```

Возможно, выбрано не самое изящное решение, но оно достаточно понятно и замечательно работает.

nbd + squashed live

На этот раз мы сэкономили ещё всего лишь 3% трафика (в пределах погрешности). Разница во времени загрузки объясняется тем, что при использовании NFS перед подключением к серверу делается принудительная пауза в 10 секунд, а в случае сервера NBD такой задержки нет.

Педаль в пол

Давайте попробуем ускорить загрузку. Самое слабое звено в нашей цепочке загрузки — TFTP сервер. Полностью исключить его мы не сможем, но минимизировать его присутствие можно с помощью загрузчика iPXE, как посоветовал kvaps в комментариях к предыдущей статье.

Подключитесь к загрузочному серверу под именем username.

Меню с вариантами загрузки мы делать не будем, а автоматически загрузимся в самый быстрый на текущий момент:

```
nano ~/myscript.ipxe
```

```
#!ipxe
```

```
ifopen net0
```

```
set server_ip 192.168.1.100
```

```
set http_path http://${server_ip}
```

```
set kern_name vmlinuz-linux
```

```
kernel ${http_path}/${kern_name} || read void
```

```
initrd ${http_path}/initramfs-linux.img || read void
```

```
imgargs ${kern_name} add_efi_memmap
```

```
ip=${net0/ip}:${server_ip}:${net0/gateway}:${net0/netmask}::eth0:none
```

```
nbd_host=${server_ip} nbd_name=habrahabr root=/dev/nbd0 || read void
```

```
boot || read void
```

Мы планируем получать файлы vmlinuz-linux и initramfs по протоколу HTTP. Внедрим наш скрипт в загрузчик:

```
sudo pacman -S git && git clone git://git.ipxe.org/ipxe.git
```

```
cd ipxe/src/
```

```
make bin/undionly.kpxe EMBED=/home/username/myscript.ipxe
```

Возвращаемся в root на сервере и копируем загрузчик:

```
cp {/home/username/ipxe/src/bin,$root/boot}/undionly.kpxe
```

Исправим DHCP сервер таким образом, чтобы он предлагал скачивать новый файл:

```
nano /etc/dhcpd.conf
```

```
#if option architecture = 7 {
```

```
# filename "/grub/x86_64-efi/core.efi";
```

```
# } else {
```

```
# filename "/grub/i386-pc/core.0";
```

```
#}
```

```
filename "/undionly.kpxe";
```

```
systemctl restart dhcpd4
```

Устанавливаем HTTP сервер:

```
pacman -S apache
```

привязываем папку с загрузчиком к рабочей папки сервера:

```
mount --bind /srv/nfs/diskless/boot/ /srv/http/
```

Можно перемонтировать в режим «только для чтения»:

```
mount -o remount,ro,bind /srv/nfs/diskless/boot/ /srv/http/
```

Запускаем сервер:

```
systemctl start httpd
```

Смотрим, что происходит на сервере:

```
vnstat -l
...
                           rx      |       tx
--------------------------------------+------------------
  bytes                    1,50 MiB  |      206,73 MiB
--------------------------------------+------------------
          max            2,96 Mbit/s  |   191,95 Mbit/s
      average          684,08 kbit/s  |    94,08 Mbit/s
          min               5 kbit/s  |        1 kbit/s
--------------------------------------+------------------
  packets                    22762  |           90737
```

```
---------------------------------------+------------------
          max                  5735 p/s  |        9871 p/s
      average                  1264 p/s  |        5040 p/s
          min                     3 p/s  |           1 p/s
---------------------------------------+------------------
  time                      18 seconds
```

Выигрыш в скорости загрузки от замены TFTP на HTTP заметен невооружённым глазом и это не единственный примечательный момент iPXE. Например, здесь показано, как можно прямо во время загрузки выбрать сервер с официальным образом установочной флешки и загрузиться в него прямо через Интернет без необходимости предварительного скачивания. Уверен, что теперь вы сможете повторить то же самое и со своим образом.

Возвращаемся на сервер

Попробуйте добавить обработчик live в наш загрузочный сервер. Сейчас правила rsync пропускают копирование содержимого /srv, где у нас находятся файлы клиента. Мы можем поменять правила или примонтировать директорию с помощью systemd:

```
nano /etc/fstab
```

```
LABEL=HABR        /    ext4 rw,relatime,data=ordered 0 1
/srv/new_root/srv /srv none bind                     0 1
```

В данном случае папки /srv/new_root/srv и /srv связываются в режиме полного доступа на чтение и запись, но мы знаем решения.

Тот факт, что загрузочный сервер может работать в режиме «только для чтения», будет весьма полезен для систем, установленных на недорогую USB флешку. С такого накопителя лучше побольше читать, и поменьше на него записывать. Если вы откроете его в интернет, то получите дополнительную степень защиты. Например, роутер

открывает защищенный VPN канал, на другом конце которого находится загрузочный сервер…

Чтобы переписать систему на флешку (с жёстким диском принцип тот же самый), её нужно вставить в компьютер и подключить к VirtualBox (Меню Устройства > Устройства USB и выбрать нужную из списка). Список доступных блочных устройств проверяется командой lsblk, как в самой первой статье. Разметьте флешку пометив загрузочной, отформатируйте с той же меткой HABR и примонтируйте к /mnt.

Создадим новый файл с правилами для rsync:

```
nano /root/clone_filter
```

```
+ /boot/*
```

```
+ /etc/*
```

```
+ /home/*
```

```
+ /srv/*
```

```
+ /usr/*
```

```
+ /var/*
```

```
- /*/*
```

Дождитесь выполнения команды:

```
rsync -aAXv /* /mnt --filter="merge /root/clone_filter"
```

В моём случае флешка — /dev/sdb

```
arch-chroot /mnt grub-install --target=i386-pc --force --recheck
```

```
/dev/sdb
```

Остаётся отмонтировать /mnt и можно загружаться из копии.

PS Решение разрабатывалось для автоматизации компьютерных классов. Система одинаково работает на пожертвованных и новых компьютерах с самыми разнообразными конфигурациями. Восстановление системы на клиенте к первоначальному состоянию производится обычной перезагрузкой. Нужные данные можно сохранять на диске загрузочного сервера или на любом другом сетевом или локальном накопителе.

Поделитесь своими идеями применения.

Теги:

- новичкам
- udev
- nbd
- nfsv4
- tftp
- ipxe
- rsync
- установка linux
- initramfs