Руководство разработчика

Сетевой шлюз на Orange Pi R1 (Rapsberry Pi 2/3) и Yocto Project

v0.7.4 (v0.7.3)

АННОТАЦИЯ

Yocto Project - это совместный Open Source проект для упрощения разработки дистрибутивов для встраиваемых систем. Yocto содержит большое количество шаблонов, метаданных и инструментальных средств сборки, и поддерживает очень большое количество аппаратных платформ.

Poky – это эталонная система сборки в рамках проекта Yocto Project. Название Poky также относится к эталонному дистрибутиву Linux, который создается этой системой сборки и может быть чрезвычайно минималистичным (core-image-minimal). В этом документе мы будем руководствоваться шаблоном core-image-minimal и на его основе создадим рецепт сборки дистрибутива для простого сетевого шлюза.

Основная плата, для которой собирается дистрибутив это «Orange Pi R1».

Примечание: дистрибутив также можно собрать для платы Raspberry Pi 2B/3B. Например в отладочных целях, но в этом случае вам понадобиться дополнительная внешняя сетевая плата USB-Ethernet.

1. *

СОДЕРЖАНИЕ

2. Установка Yocto Project в Ubuntu

Для сборки дистрибутива с помощью Yocto Project в Ubuntu вам необходимо установить следующие пакеты:

```
sudo apt-get install -y --no-install-suggests --no-install-recommends \
gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
xterm
```

Пакеты устанавливаются с помощью команды **apt-get install** и команды повышения привилегий - **sudo**. В системе Ubuntu это широко распространенная практика, когда для выполнения административных действий используется команда sudo (при создании основного пользователя системы, он автоматически прописывается в группу «sudo»). Более подробную инструкцию по установке вы можете посмотреть здесь: https://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#ubuntu-packages

3. Назначение проекта сетевого шлюза

Проект предназначен для самостоятельной сборки с «нуля» Linux дистрибутива для одноплатного компьютера **«Orange Pi R1»** с минимальным набором программ, достаточным для использования Orange Pi R1 в качестве простого шлюза.

Шлюз это компьютер, который подключен как минимум к двум разным подсетям, и который умеет передавать пакеты из одной подсети в другую.

Основная функция нашего «Шлюза» - фильтрация внешнего (интернет) сетевого трафика приходящего на один сетевой интерфейс и передача его после фильтрации на другой сетевой интерфейс связанный с локальной сетью т.е. шлюз выполняет функцию **«Брандмауэра»** для защиты внутренней сети. Также шлюз будет обеспечивать подключение пользователей внутренней сети к ресурсам Интернет от имени своего ip адреса, т.е обеспечивать так называемую трансляцию сетевых адресов NAT (Network Address Translation).

Еще одной функцией «Шлюза» будет возможность получения от него динамических ip адресов, т.е. любая новая машина в локальной сети с помощью широковещательного запроса сможет получить ip адрес из выделенного диапазона адресов.

Таким образом «Шлюз» будет выполнять роль DHCP сервера (предполагается что шлюз одним сетевым интерфейсом подключен к сетевому коммутатору локальной сети).

3.1. Репозитории проекта

– для платы Orange Pi R1 (версия v0.7.4):

```
# xml файл конфигурации => orange/hardknott/router-bs-0.7.4.xml
git clone https://github.com/berserktv/bs-manifest.git

# слой для Yocto Project
git clone https://github.com/berserktv/router-bs.git -b opi-hardknott
```

– для платы Raspberry Pi 2B/3B (версия v0.7.3):

```
# xml файл конфигурации => raspberry/gatesgarth/router-bs-0.7.3.xml
git clone https://github.com/berserktv/bs-manifest.git

# слой для Yocto Project
git clone https://github.com/berserktv/router-bs.git -b rpi-gatesgarth
```

3.2. Требования к оборудованию

Для создания сетевого шлюза, нам понадобиться одноплатный компьютер **«Orange Pi R1»** - на базе SoC Allwinner H2+ с характеристиками:

– 4 ядра ARM Cortex-A7 с тактовой частотой 1.2 ГГц;

– объём оперативной памяти — 256 Мбайт DDR3 SDRAM;

– на плате размещены два сетевых интерфейса;

– корпус;

– карта памяти - microSDHC, любого размера начиная от 8Гб.

Примечание. просто меньшего размера сейчас трудно найти.

4. Управление слоями Yocto с помощью Repo

Yocto Project может использовать большое количество слоев от разных поставщиков - разработчиков оборудования, и всем этим необходимо как то управлять. Представьте себе, что у вас есть десяток различных плат, и к каждой плате поставляется отдельный BSP гит репозиторий, и это не считая инфраструктуры самого Yocto проекта, плюс возможная дополнительная функциональность из OpenEmbedded.

В такой ситуации отдельным простым скриптом установки уже не отделаешься. Волей, неволей приходиться искать инструменты, которые умеют это делать хорошо. Даже более чем хорошо. Одним из лучших инструментов такого рода является утилита Google - Repo.

Repo - это основной инструмент для управления GIT репозиториями при сборке операционной системы «Андроид» с его большой кодовой базой. Repo позволяет в одном проекте управлять десятком, если не сотней отдельных гит репозиториев, версии которых вы можете аккуратно указать в одном xml файле Манифеста и для правильной синхронизации всех версий всех репозиториев вам достаточно выполнить одну команду:

```
repo sync
```

4.1. Установка Repo

С помощью следующего набора команд вы можете установить Repo в ваш домашний каталог ~/bin (команду curl можно установить отдельно: sudo apt-get install curl):

```
mkdir ~/bin
curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

Примечание. в ОС Ubuntu каталог HOME/bin должен добавляться с стандартный путь запуска автоматически, вы можете в этом убедиться проверив файл HOME/.profile пример наличие строки: PATH=”$HOME/bin:$HOME/.local/bin:$PATH”

4.2. Манифест для сборки дистрибутива

Для управления разными версиями дистрибутива используется проект https://github.com/berserktv/bs-manifest

4.2.1. Структура bs-manifest

```
├── COPYING.MIT
├── orange
│   ├── hardknott
│   │   ├── router-bs
│   │   │   └── conf
│   │   │       ├── bblayers.conf
│   │   │       └── local.conf
│   │   └── router-bs-0.7.4.xml
│   └── shell-opi-zero.sh
├── README.md
└── setup-environment
```

Примечание. Структура показанная выше касается сборки проекта router-bs для платы Orange Pi R1, используется ветка Yocto Project => hardknott. Для инициализации слоев сборки и их загрузки необходимо выполнить следующие команды:

```
PATH=${PATH}:~/bin
mkdir router-bs
cd router-bs
repo init -u https://github.com/berserktv/bs-manifest -m orange/hardknott/router-bs-0.7.4.xml
repo sync
```

4.2.2. Содержимое xml конфигурации манифеста (router-bs-0.7.4.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <default sync-j="4" revision="hardknott"/>
  <remote fetch="https://git.yoctoproject.org/git" name="yocto"/>
  <remote fetch="https://github.com/openembedded" name="oe"/>
  <remote fetch="https://github.com/linux-sunxi" name="sunxi"/>
  <remote fetch="https://github.com/berserktv" name="bs"/>

  <project remote="bs" revision="master" name="bs-manifest" path="sources/bs-manifest">
    <linkfile dest="setup-environment" src="setup-environment"/>
    <linkfile dest="shell.sh" src="orange/shell-opi-zero.sh"/>
    <linkfile dest="sources/base" src="orange/hardknott/router-bs"/>
  </project>

  <project remote="yocto" revision="hardknott" name="poky" path="sources/poky"/>
  <project remote="oe"    revision="hardknott" name="meta-openembedded" path="sources/meta-openembedded"/>
  <project remote="sunxi" revision="hardknott" name="meta-sunxi"  path="sources/meta-sunxi"/>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  <project remote="bs"    revision="opi-hardknott" name="router-bs" path="sources/router-bs"/>
</manifest>
```

4.2.3. Описание Манифеста

– в начале манифеста тегами remote обозначены два основных GIT репозитория и один вспомогательный:

- https://git.yoctoproject.org/git - *Yocto* репозиторий, именованный как yocto;
- https://github.com/openembedded - *OpenEmbedded* репозиторий, именованный как oe;
- https://github.com/berserktv - вспомогательный GIT репозиторий, именованный как bs.

– в следующей части манифеста с помощью сокращенных именований осуществляется работа с проектами расположенными в этих репозиториях, тег project содержит следующие атрибуты

- remote - имя удаленного именованного репозитория;
- revision - название ветки или hash версии;
- name - имя проекта в указанном репозитории;
- path - локальный путь проекта в вашей файловой системе.

Пример работы тега:

```
<project remote="bs" revision="master" name="bs-manifest" path="sources/bs-manifest">
</project>

данный xml тег описывает выполнение следующей команды:
git clone https://github.com/berserktv/bs-manifest -b master sources/bs-manifest
```

В теле тега project указываются команды создания символических ссылок на нужную инфраструктуру вспомогательных скриптов начальной инициализации и штатного запуска системы сборки Poky.

Пример указания тегов linkfile:

```
<project remote="bs" revision="master" name="bs-manifest" path="sources/bs-manifest">
    <linkfile dest="setup-environment" src="setup-environment"/>
    <linkfile dest="shell.sh" src="orange/shell-opi-zero.sh"/>
    <linkfile dest="sources/base" src="orange/hardknott/router-bs"/>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
</project>

Создание символической ссылки разворачивается в следующую команду: ln -s src dest т.е.
# создание двух ссылок на файлы
ln -s sources/bs-manifest/setup-environment  setup-environment
ln -s sources/bs-manifest/orange/shell-opi-zero.sh shell.sh

# создание ссылки на каталог, разворачивается в следующий набор команд
cd sources
ln -s bs-manifest/orange/hardknott/router-bs base
```

4.2.4. Инициализация переменных Poky

Скрипт начальной инициализации setup-environment был взят из проекта Freescale Community. Скрипт отвечает за стартовую инициализацию переменных системы сборки Poky, скрипт создает структуру каталогов, в которой очень хорошо разделяется:

– **build**: каталог сборки;

– **source**: исходный код рецептов сборки;

– **download**: каталог загрузки кода программ (git базы, tar.gz архивы).

Содержимое скрипта setup-environment можно посмотреть здесь:

https://github.com/berserktv/bs-manifest/blob/master/setup-environment

Содержимое скрипта shell.sh:

```
#!/usr/bin/env bash

MACHINE='orange-pi-zero' source ./setup-environment build

echo "you may try 'bitbake core-image-minimal'"
bash
```

Этот корневой скрипт служит для начальной конфигурации переменных среды сборки, он вызывается в начале сеанса работы.

4.2.5. Инициализация и синхронизация слоев сборки с помощью Repo

Для инициализации repo необходимо выполнить команды:

```
$  PATH=${PATH}:~/bin
$  mkdir router-bs
$  cd router-bs
$  repo init -u https://github.com/berserktv/bs-manifest -m orange/hardknott/router-bs-0.7.4.xml
$  repo sync
```

– где -u https://github.com/berserktv/bs-manifest указывает GIT путь к проекту манифеста;

Примечание. можно еще указать -b имя_ветки , если ключ -b не указывать, то подразумевается ветка master (по умолчанию);

– где путь -m **orange/hardknott/router-bs-0.7.4.xml** к конфигурационному файлу указывает следующее:

– имя аппаратной платформы для которой осуществляется сборка — **orange**;

– имя основной рабочей ветки Yocto/OpenEmbedded — **hardknott**;

– кодовое имя версии (название сборки) — **router-bs**;

– и наконец цифровой номер версии которая собирается — **0.7.4**.

Создание конфигурации Yocto Project

После того, как будет выполнена команда repo sync, можно приступать к созданию основной конфигурации Yocto Project:

```
./shell.sh
```

после завершения работы скрипта, будет создан каталог build/conf с двумя основными файлами:

– local.conf - управляющие переменные сборки с названием платформы, типом дистрибутива, типом пакетов сборки и т.д;

– bblayers.conf - конфигурация подключенных слоев Yocto Project.

Примечание. по умолчанию скрипт setup-environment ищет по пути sources/base/conf начальную конфигурацию, и если файлы local.conf и bblayers.conf существуют, то они копируются в build/conf (см. переменную TEMPLATES в setup-environment) т.е. файлы берутся из

sources/bs-manifest/orange/hardknott/router-bs/conf

4.2.6. Конфигурационный файл build/conf/local.conf

```
MACHINE ??= 'orange-pi-zero'
DISTRO ?= 'poky'
PACKAGE_CLASSES ?= "package_deb"
EXTRA_IMAGE_FEATURES ?= "debug-tweaks"
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS = "\
    STOPTASKS,${TMPDIR},1G,100K \
    STOPTASKS,${DL_DIR},1G,100K \
    STOPTASKS,${SSTATE_DIR},1G,100K \
    STOPTASKS,/tmp,100M,100K \
    ABORT,${TMPDIR},100M,1K \
    ABORT,${DL_DIR},100M,1K \
    ABORT,${SSTATE_DIR},100M,1K \
    ABORT,/tmp,10M,1K"
PACKAGECONFIG_append_pn-qemu-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
CONF_VERSION = "1"

DL_DIR ?= "${BSPDIR}/downloads/"

# for libs: "mpeg2dec libmad ffmpeg x264"
LICENSE_FLAGS_WHITELIST += "commercial"
```

основные переменные файла local.conf:

– MACHINE: название платформы под которую осуществляется сборка;

– DISTRO: название категории дистрибутива;

– PACKAGE_CLASSES: формат пакетов для установки ПО;

– LICENSE_FLAGS_WHITELIST: использование дополнительных лицензий.

дополнительные настройки для семейства плат Raspberry Pi:

```
GPU_MEM = "128" - количество видео памяти для видео адаптера GPU (выделяется из ОЗУ)
GPU_MEM_256 = "112" - то же самое только для плат с общим размером ОЗУ = 256Мб
GPU_MEM_512 = "160" - то же самое только для плат с общим размером ОЗУ = 512Мб
GPU_MEM_1024 = "320" - то же самое только для плат с общим размером ОЗУ = 1024Мб
```

Примечание. например если оставить только переменную GPU_MEM = «128», то для всех плат RPI, RPI2, RPI3 не зависимо от количества реальной оперативной памяти на плате будет всегда выделено для GPU - 128Мб (и общий размер ОЗУ уменьшиться на это значение), в случае указания всех переменных, директивы GPU_MEM_256, GPU_MEM_512, GPU_MEM_1024 являются более приоритетными;

4.2.7. Конфигурационный файл build/conf/bblayers.conf

```
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = "6"
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) + '/../..')}"
BBFILES ?= ""

BBLAYERS ?= " \
  ${BSPDIR}/sources/poky/meta \
  ${BSPDIR}/sources/poky/meta-poky \
  ${BSPDIR}/sources/poky/meta-yocto-bsp \
  ${BSPDIR}/sources/meta-openembedded/meta-oe \
  ${BSPDIR}/sources/meta-sunxi \
  ${BSPDIR}/sources/router-bs/meta-router-bs \
  "
```

Для сборки сетевого шлюза, помимо штатных слоев Yocto:

```
${BSPDIR}/sources/poky/meta \
${BSPDIR}/sources/poky/meta-poky \
${BSPDIR}/sources/poky/meta-yocto-bsp \
```

подключен слой с дополнительной функциональностью из OpenEmbedded:

```
${BSPDIR}/sources/meta-openembedded/meta-oe \
```

затем подключен основной BSP слой для платформы Orange Pi:

```
${BSPDIR}/sources/meta-sunxi \
```

и в конце подключен наш слой meta-router-bs, с функциональностью шлюза:

```
${BSPDIR}/sources/router-bs/meta-router-bs \
```

5. Механизм сборки дистрибутива в Yocto Project

В Yocto Project сборка каждой программы описывается с помощью рецепта сборки. Основным языком описания рецепта является **«bash»** с возможностью вставок частей кода на языке **«python»**.

Основную информацию по синтаксису вы можете почерпнуть из руководства Yocto Project: http://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html

Набор рецептов сборки в зависимости от назначения можно объединять в отдельные слои сборки. В нашем случае все дополнительные рецепты которые расширяют функциональность базового дистрибутива Yocto, с минимальным количеством пакетов - «core-image-minimal» и привносят в него нужную нам функциональность **«сетевого шлюза»** будут храниться в слое «meta-router-bs».

Слой «meta-router-bs» прописывается в конфигурационном файле build/conf/bblayers.conf в момент установки Poky.

5.1. Структура слоя «meta-router-bs»

```
├── conf
│   └── layer.conf
├── recipes-connectivity
│   ├── dhcp
│   └── shorewall
├── recipes-core
│   ├── psplash
│   └── systemd
├── recipes-kernel
│   └── linux
└── recipes-router-bs
    └── images
```

Для внесения нужной нам функциональности мы по возможности будем использовать так называемые дополнения для рецептов, которые располагаются в файлах с расширением .bbappend. В файле .bbappend вы можете добавить собственные вызовы команд для штатного метода рецепта сборки, например в метод do_install, do_configure, do_compile и т.д.

В слое нашего шлюза находятся:

- **conf**: конфигурация слоя;
- **recipes-connectivity**: рецепты сборки сетевых сервисов;
- **recipes-core**: базовые репепты, настройка сетевых интерфейсов и заставка;
- **recipes-kernel**: рецепт относящийся к сборки linux ядра;
- **recipes-router-bs**: рецепт сборки прошивки с функциональностью шлюза.

5.2. Включение подсистемы инициализации Systemd в Yocto

По умолчанию в Yocto Project для управления службами в Linux используется подсистема инициализации Sysvinit. В настоящий момент эта система является устаревшей и в большинстве случаев уже не используется. Ей на смену пришла система инициализации Systemd.

Для того, чтобы включить подсистему инициализации systemd в конфигурационном файле слоя **meta-router-bs/conf/layer.conf** добавляются строки:

```
DISTRO_FEATURES_append = " systemd "
DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"
VIRTUAL-RUNTIME_init_manager = "systemd"
VIRTUAL-RUNTIME_initscripts = ""
```

Переменная DISTRO_FEATURES включает указанные функции дистрибутива, это наборы предопределенных групп пакетов, которые удобно подключать с помощью одной переменной. На уровне рецептов такая функциональность включается следующим образом (пример использования):

```
PACKAGECONFIG ??= "${@bb.utils.contains('DISTRO_FEATURES', 'x11', 'x11', '', d)} \
                   ${@bb.utils.contains('DISTRO_FEATURES', 'wayland', 'wayland', '', d)}"

PACKAGECONFIG[wayland] = "EGL_TYPE=framebuffer,,,"
PACKAGECONFIG[x11] = "EGL_TYPE=x11,,virtual/libx11 libxau libxdmcp libdri2,"
```

Переменная PACKAGECONFIG предоставляет средство включения/отключения функций рецепта. Первая переменная определяет список функций, а последующие переменные вида PACKAGECONFIG[<function>] определяют поведение функций.

В поведении функций можно предоставить до четырех зависимых от позиции аргументов, разделенных запятыми:

1. Дополнительные аргументы, которые должны быть добавлены в список аргументов конфигурационного скрипта (переменная EXTRA_OECONF) при включении функции;

2. Дополнительные аргументы, которые должны быть добавлены в EXTRA_OECONF при отключении функции;

3. Дополнительные зависимости на время сборки, которые нужно добавить в DEPENDS при включении функции;

4. Дополнительные установленные в систему зависимости, которые нужно добавить в RDEPENDS при включении функции.

5.3. Пакет поддержки платформы «Orange Pi»

Для того, чтобы собрать дистрибутив для Orange Pi Zero в Yocto Project существует отдельный слой - meta-sunxi. На github он располагается по адресу: https://github.com/linux-sunxi/meta-sunxi

Плата которую мы выбрали - «Orange Pi R1» идентична плате «Orange Pi Zero» (за исключением дополнительной встроенной сетевой платы), и поэтому для сборки прошивки под нее мы будем указывать платформу MACHINE = «orange-pi-zero»

Слой «meta-sunxi» хорошо работает прямо из коробки, это значит что нам пока без надобности знать особенности аппаратной работы и тонкости конфигурации Orange Pi Zero (все настройки можно оставить по умолчанию), т.е. нам достаточно подключить этот «BSP» слой в «Poky», и после сборки образа аппаратная поддержка платы Orange Pi R1 (Orange Pi Zero) будет включена в наш дистрибутив «router-bs».

5.4. Поддержка USB сетевых плат для «Orange Pi Zero»

Конечно в идеале нам необходимо взять плату с двумя встроенными сетевыми интерфейсами Orange Pi R1, и далее это главу вы можете пропустить. Если же по какой то причине у вас есть под рукой только плата Orange Pi Zero, то рассмотрим гипотетический вариант использования второй, внешней USB => Ethernet сетевой платы. Таким же образом в дальнейшем вы можете подключить любое дополнительное оборудование (ну если конечно в Linux ядре это оборудование уже поддерживается).

Для корректной работы сетевой платы в Linux нам потребуется две вещи:

– Поддержка Linux ядром определенной сетевой платы;

– Наличие в системе модуля ядра для определенной сетевой платы.

Возьмем к примеру сетевую плату Readyon RD-KY88772A, скорости подключения до 100 Мб/сек нам достаточно. Найдем чипсет на котором плата работает - это AX88772A разработка фирмы «ASIX».

Теперь найдем название параметра в конфигурации ядра отвечающую за драйвер чипсета AX88772A, лучше всего поискать сочетание слов «AX88772A cateee.net» (на google.com) где «cateee.net» - сайт с полным описанием конфигураций модулей ядра Linux.

Сразу находим название конфигурации ядра - CONFIG_USB_NET_AX8817X и название нужного нам модуля ядра asix - https://cateee.net/lkddb/web-lkddb/USB_NET_AX8817X.html

Далее нам необходимо удостовериться, что в той версии ядра linux которое мы будет собирать в рамках Yocto Project, нужная нам конфигурация поддерживается, для этого мы можем в начале собрать ядро командой «bitbake linux-mainline» см. Краткую инструкцию по созданию прошивки в конце документа, а затем посмотреть сохраненный файл с текущей конфигурацией ядра:

```
build/tmp/work/orange_pi_zero-poky-linux-gnueabi/linux-mainline/5.4.69-r0/defconfig
```

В нашем случае поддержка сетевой платы отсутствует, потому что нет строки CONFIG_USB_NET_AX8817X и мы можем добавить эту поддержку с помощью включения фрагментов конфигураций ядра (см. следующую главу).

Для включения в образ поддерживаемых модулей ядра описанных в defconfig, в файле рецепта router-bs-image.bb мы можем указать пакет «kernel-modules». Это добавит все собранные модули в наш образ, увеличив его ~ на 28 Мбайт, но позволит избавиться от части проблем с оборудованием.

Альтернативным вариантом является явное указание названия модуля, у нас это «kernel-module-asix»

Примечание. Также следует отметить, что окончательная конфигурация ядра после сборки будет находиться здесь:

```
build/tmp/work/orange_pi_zero-poky-linux-gnueabi/linux-mainline/5.4.69-r0/build/.config
```

и если какое то подключенное устройство у вас не работает, то проверьте файл более внимательно (в defconfig содержаться рекомендуемые параметры и в окончательную конфигурацию, параметр может не попасть, если он не прошел проверку на совместимость с текущей версией ядра, проверка осуществляется автоматически, во время сборки ядра).

5.5. Механизм поддержки фрагментов конфигураций ядра

В Yocto Project вы можете создать небольшой конфигурационный файл, в котором указать только те параметры ядра, которые вам необходимы, и эти параметры будут добавлены к параметрам по умолчанию, которые уже входят в ядро, которое вы собираете:

исходный конфигурационный файл настройки ядра находится в рецептах слоя meta-sunxi:

```
sources/meta-sunxi/recipes-kernel/linux/linux-mainline/arm/defconfig
```

файл с дополнительными параметрами конфигурации **router-extra.cfg**:

```
# This option adds support for ASIX AX88xxx based USB 2.0 10/100 Ethernet adapters.
CONFIG_USB_NET_AX8817X=m
# Ethernet adapters, driver for TP Link UE300
CONFIG_USB_RTL8152=m

##### Orange Pi Zero ####
# This driver is used for H3/A83T/A64 EMAC ethernet controller, name module dwmac-sun8i
CONFIG_DWMAC_SUN8I=m
```

Файл будет располагаться по следующему пути:

```
meta-router-bs/recipes-kernel/linux/files/router-extra.cfg
```

Чтобы подключить фрагмент конфигураций для нашей сборки ядра нам необходимо добавить дополнение для рецепта сборки:

```
meta-router-bs/recipes-kernel/linux/linux-mainline_%.bbappend
```

следующего содержания:

```
# дополнительный параметры конфигурации описываются в router-extra.cfg
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://router-extra.cfg"

# в BSP слое meta-raspberrypi не работают фрагменты конфигураций
# https://github.com/agherzan/meta-raspberrypi/issues/14
# поэтому делаю напрямую
# в методе do_kernel_configme конфигурация ядра копируется из базы архитектур arch/ в рабочий каталог
do_kernel_configme_append() {
    cat ${WORKDIR}/router-extra.cfg >> ${WORKDIR}/defconfig
}
```

6. Настройка сетевого интерфейса шлюза

Для работы устройства в качестве шлюза, вам необходимо настроить два сетевых интерфейса. На конечном устройстве настройки будут храниться в файлах:

– настройка для первого сетевого интерфейса eth0, который будет считаться внешним (открытым из Интернета):

/etc/systemd/network/20-eth0.network

– настройка для второго сетевого интерфейса eth1, который будет считаться внутренним:

/etc/systemd/network/20-eth1.network

Содержимое файла **meta-router-bs/recipes-core/systemd/files/20-eth0.network**:

```
[Match]
Name=eth0

[Network]
Address=192.168.0.1/24
###Gateway=
# ip address DNS servers - OpenDNS Family Shield
DNS=208.67.222.123 208.67.220.123
```

Здесь представлена настройка IP-адреса шлюза по умолчанию => 192.168.0.1 для маски подсети /24 т.е. 255.255.255.0.

В качестве адресов DNS у нас указаны бесплатные общедоступные серверы OpenDNS, с фильтрацией «взрослого содержимого».

Все параметры конфигурации для внешнего сетевого интерфейса вы должны взять у своего «Интернет Провайдера» т.е. компании поставщика услуг на доступ в Интернет.

Содержимое файла **meta-router-bs/recipes-core/systemd/files/20-eth1.network**:

```
[Match]
Name=eth1

[Network]
Address=10.0.8.1/24
###Gateway=
# ip address DNS servers - OpenDNS Family Shield
DNS=208.67.222.123 208.67.220.123
```

Указаны не маршрутизируемые (внутренние адреса) для локальной подсети 10.0.8.0 и маска /24 т.е. 255.255.255.0 (эти настройки вы можете не менять).

Соответствие масок подсетей можно посмотреть здесь.

7. Настройка DHCP сервера на шлюзе

Наш «Шлюз» будет выполнять роль DHCP сервера (протокол динамической конфигурации узла). Настройки DHCP сервера на конечном устройстве будут храниться в файле «/etc/kea/kea-dhcp4.conf»

Содержимое файла (фрагмент конфигурации) «**kea-dhcp4.conf**»

```
{
"Dhcp4": {
    // Add names of your network interfaces to listen on.
    "interfaces-config": {
        "interfaces": [ "eth1" ]
    },

    "control-socket": {
        "socket-type": "unix",
        "socket-name": "/tmp/kea4-ctrl-socket"
    },

    "lease-database": {
        "type": "memfile",
        "lfc-interval": 3600
    },

    "expired-leases-processing": {
        "reclaim-timer-wait-time": 10,
        "flush-reclaimed-timer-wait-time": 25,
        "hold-reclaimed-time": 3600,
        "max-reclaim-leases": 100,
        "max-reclaim-time": 250,
        "unwarned-reclaim-cycles": 5
    },

    "renew-timer": 900,
    "rebind-timer": 1800,
    "valid-lifetime": 3600,

    "option-data": [
        // ip address OpenDNS Family Shield => 208.67.222.123, 208.67.220.123"
        {
            "name": "domain-name-servers",
            "data": "208.67.222.123, 208.67.220.123"
        },
    ],
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    "subnet4": [
        {
            "subnet": "10.0.8.0/24",
            "pools": [ { "pool": "10.0.8.14 - 10.0.8.100" } ],

            "option-data": [
                {
                    // For each IPv4 subnet you most likely need to specify at
                    // least one router.
                    "name": "routers",
                    "data": "10.0.8.1"
                }
            ]
        }
    ]
}
}
```

Документацию на сервер вы можете посмотреть здесь:

https://kea.readthedocs.io/en/kea-2.0.0

Укажем лишь основные настройки:

1. Интерфейс или адрес через который будет работать dhcp4: **interfaces-config**;
2. Расположение хранения базы lease: **lease-database**;
3. Временные настройки: **expired-leases-processing**;
4. Основные DNS сервера: Dhcp4 => option-data => **domain-name-servers**;
5. Настройка — подсети, диапазоны адресов (пулы) и шлюз по умолчанию: **subnet4**;

Итак мы указали основной минимальный набор параметров, которые необходимы для работы DNS сервера **KEA** на нашем «Шлюзе».

8. Фильтрация сетевых пакетов - функции «Брандмауэра»

В linux фильтрация пакетов происходит на уровне ядра операционной системы в модуле «netfilter» и управляется утилитой iptables.

Так как правила настройки iptables крайне сложные и запутанные мы будем использовать, удобную утилиту, которая позволяет в простой пользовательской форме задать эти правила, без необходимости изучения «iptables». Для настройки будет использована программа скрипт, программа Shorewall.

Shorewall использует правила описания каждого из аспектов фильтрации в отдельном файле, и при запуске использует Perl для разбора файлов и генерации правил уже в формате «iptables».

8.1. Конфигурация «Зоны» шлюза

Правила хождения пакетов в Shorewall используют абстракцию зон — все сети, которые вы хотите обрабатывать, надо как-то обозвать. Для начала следует определиться с названиями зон. В нашем случае их будет 3:

– **fw**: необходимая зона, содержащая сам файрволл;

– **net**: провайдерская сеть;

– **loc**: локальная (или домашняя) сеть с DHCP.

Конфигурация получается примерно такой, файл zones:

– путь на целевом устройстве: /etc/shorewall/zones;

– путь в слое meta-router-bs/recipes-connectivity/shorewall/files/zones.

```
...
fw    firewall
net   ipv4
loc   ipv4
```

8.2. Интерфейсы и хосты

После описание зон, необходимо распределить их между сетевыми интерфейсами, файл interfaces:

– путь на целевом устройстве: /etc/shorewall/interfaces;

– путь в слое meta-router-bs/recipes-connectivity/shorewall/files/interfaces.

```
...
#ZONE INTERFACE BROADCAST OPTIONS
net     eth0    detect    tcpflags,nosmurfs,routefilter,logmartians
loc     eth1    detect    dhcp,tcpflags,nosmurfs,routefilter,logmartians
```

– Параметр «detect» говорит shorewall автоматически определить широковещательный адрес, его можно прописать и вручную, например так: 10.0.8.255 для внутреннего сетевого интерфейса с ip адресом 10.0.8.1, см. выше;

– Опция tcpflags - пакеты, поступающие на этот интерфейс проверяются на наличие определенных незаконных комбинаций TCP флагов;

– Опция routeback - заставляет весь трафик, пришедший через интерфейс X, возвращаться к отправителю через него же (без этого практически не обойтись в ситуации, когда провайдеров несколько);

– Опции nosmurfs,logmartians - разные по своему действию, но близкие по духу параметры, заставляющие Shorewall обращать внимание на пакеты, которых «точно не должно быть на этом интерфейсе». Так, пакеты с широковещательным адресом отправителя отклоняются (nosmurfs), «марсиане» (пакеты с некорректным исходным адресом) протоколируются (logmartians);

– Опция routefilter - предписывает включить в ядре фильтрацию по маршруту, но пользоваться им следует с осторожностью: в схеме с несколькими исходящими каналами возможны трудно диагностируемые неполадки;

– Опция dhcp - сообщает, что на интерфейсе выполняется DHCP-клиент или сервер.

8.3. Политики

После разделения окружающего мира на зоны, обязательно нужно определить политики т.е. набор правил которые действуют для пакетов, самая правильная политика по умолчанию «все запрещено», а то что разрешено явно указывается в виде отдельных правил.

Необходимо определить политики для всех возможных направлений пробегания трафика. Лучше всего в конец дописать дефолтную политику, а чуть выше — исключения: переменная FW является служебной и содержит имя зоны брандмауэра.

– путь на целевом устройстве /etc/shorewall/policy;

– путь в слое meta-router-bs/recipes-connectivity/shorewall/files/policy.

```
...
#SOURCE   DEST    POLICY   LOG LEVEL LIMIT:BURST
# разрешение трафика из зоны локальной сети во внешнюю
loc       net     ACCEPT

# запрещение трафика инициированное внешней сетью
net       all     DROP     info

# отвергать все другие запросы на соединение
# (Shorewall требует наличия такой политики,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# применимой для всех остальных запросов).
# строка обязательно должна быть последней в файле policy
all       all     REJECT   info
```

8.4. Настройка трансляции сетевых адресов NAT

Для того, чтобы все компьютеры из локальной сети могли обращаться в интернет и получать ответ, нам необходимо настроить «Маскарадинг», это когда все запросы в интернет отправляются от имени нашего маршрутизатора, а когда приходит ответ, маршрутизатор автоматически понимает для какого компьютера в локальной сети он предназначается.

– путь на целевом устройстве /etc/shorewall/masq;

– путь в слое meta-router-bs/recipes-connectivity/shorewall/files/masq.

```
...
#INTERFACE          SOURCE          ADDRESS         PROTO   PORT(S) IPSEC   MARK
eth0          eth1
```

В колонке INTERFACE указывается интерфейс назначения т.е. внешний интерфейс подключенный к интернету, в нашем случае это eth0. В поле SOURCE перечисляются адреса (исходящие адреса), для которых нужно применить SNAT-преобразование: здесь, как и во многих реальных случаях, это просто eth1, то есть вся локальная сеть целиком. При желании, имя интерфейса можно заменить явным указанием «серых» адресов (скажем, 192.168.0.0/24). Запретить SNAT для избранных узлов сети можно при помощи следующего синтаксиса:

eth1:!192.168.0.1,192.168.0.3

Кстати, он действует и во многих других конфигурационных файлах Shorewall.

8.5. Настройка правил фильтрации

Для правильной настройки Shorewall используются определенные правила фильтрации, правила обычно описывают для каких протоколов и портов необходимо разрешить входящий (а иногда исходящий) трафик.

– путь на целевом устройстве /etc/shorewall/rules;

– путь в слое meta-router-bs/recipes-connectivity/shorewall/files/rules.

```
#ACTION  SOURCE DEST  PROTO DEST SOURCE ORIGINAL RATE USER/ MARK
SECTION NEW

# запрещать некорректные пакеты из внешней сети
Invalid(DROP)       net             all

# Разрешать соединения по протоколу DNS от роутера во внешнюю сеть
# иначе вы не сможете определить ip адрес узла по имени сайта
DNS(ACCEPT) $FW             net

# Разрешать соединение с роутером по протоколу SSH
# из локальной сети для администрирования
SSH(ACCEPT) loc             $FW

# Разрешить работу утилиты ping
# из локальной сети до роутера
Ping(ACCEPT)        loc             $FW

# Запретить возможность ping(ования) из внешней сети
Ping(DROP)  net             $FW

# разрешить передачу пакетов icmp от роутера
# и в локальную и во внешнюю сеть
ACCEPT              $FW             loc             icmp
ACCEPT              $FW             net             icmp

# разрешить передачу пакетов NTP (протокол сетевого времени)
# и в локальную и во внешнюю сеть
NTP(ACCEPT) loc             $FW
NTP(ACCEPT) $FW             net
```

В начале указывается действие (РАЗРЕШЕНИЕ/ЗАПРЕЩЕНИЕ), далее исходящий интерфейс, далее сетевой интерфейс назначения, далее протокол и после этого возможно явно указать номер(а) портов, для которых предназначено данное действие.

Так же по причине безопасности роутера, не рекомендуем вам оставлять возможность удаленного администрирования роутера из Интернета, пример правила: «SSH(ACCEPT) net $FW» в этом случае как минимум необходимо подобрать сложный пароль администратора (root), а также не использовать порт 22 по умолчанию, так как в любой открытой сети всегда приходят пакеты по протоколу ssh с попыткой автоматического подбора пароля.

Такой перебор выполняется специальными программами сканерами в автоматическом режиме, и в 95% случаев достаточно изменить номер стандартного порта, что бы случайным образом не оказаться взломанным

Настоятельно не рекомендуем заниматься перенастройкой Shorewall подключившись извне, как как при задании неверных правил и последующей перезагрузкой Shorewall, можно добиться блокирования собственного доступа

8.6. Остановка Shorewall

Когда файервол останавливается, маршрутизация разрешается на те хосты, которые указаны в файле routestopped:

– путь на целевом устройстве /etc/shorewall/routestopped;

– путь в слое meta-router-bs/recipes-connectivity/shorewall/files/routestopped.

```
#INTERFACE  HOST(S)                OPTIONS
eth1                -
```

В данном случае мы указываем что после остановки Shorewall разрешена маршрутизация на любые адреса («-» значит «0.0.0.0/0») с внутреннего сетевого интерфейса

ПРИМЕЧАНИЕ: если вы хотите полностью очистить правила внесенные Shorewall в «netfilter», вы должны выполнить команду «shorewall clear»

8.7. Конфигурация Shorewall

Основной конфигурационный файл расположен:

– путь на целевом устройстве: /etc/shorewall/shorewall.conf;

– путь в слое meta-router-bs/recipes-connectivity/shorewall/files/shorewall.conf.

В этом файле лежит конфигурация парсера, некоторые особенности поведения. За основу был взять конфигурационный файл из примера настройки Shorewall, файл является достаточно большим и подробно вы можете изучить его самостоятельно. Пример конфигурационного файла на целевом устройстве /usr/share/shorewall/configfiles/shorewall.conf

8.8. Примеры настройки Shorewall

В пакет поставки Shorewall входят хорошие, достаточно универсальные примеры настройки для конфигурации сетевых интерфейсов:

```
однопортовой      /usr/share/doc/shorewall/examples/one-interface
двухпортовой      /usr/share/doc/shorewall/examples/two-interfaces
трех портовой     /usr/share/doc/shorewall/examples/three-interfaces
и универсальной   /usr/share/doc/shorewall/examples/Universal
```

Примечание. путь к конфигурациям указан на целевом устройстве или же вы можете посмотреть примеры после сборки прошивки на компьютере: build/tmp/work/armv7vet2hf-neon-poky-linux-gnueabi/shorewall/4.4.27-r0.0/shorewall-4.4.27.3/Samples

Более полную документацию на русском языке по настройке Shorewall вы можете посмотреть здесь:

Основной файервол (two-interfaces)	https://shorewall.org/two-interface_ru.html
Файервол с тремя интерфейсами	https://shorewall.org/three-interface_ru.html

9. Рецепт сборки Shorewall для слоя meta-router-bs

Для того, чтобы собрать пакет Shorewall в нашем дистрибутиве, мы возьмем его рецепт сборки из проекта «OpenEmbedded»

OpenEmbedded — инфраструктура для сборки пакетов для встраиваемого Linux. OpenEmbedded предлагает решение в классе сред для кросс-компиляции. Он позволяет разработчикам создавать целостные дистрибутивы Linux для встраиваемых систем.

Исходный рецепт можно взять вот отсюда - http://git.openembedded.org/openembedded/plain/recipes/shorewall/shorewall_4.4.14.bb

И в дальнейшем модифицируем его, обновив до последней стабильной ревизии в версии 4.4, у нас это 4.4.27

Добавим рецепт shorewall_4.4.27.bb в наш слой сборки «meta-router-bs» предварительно немного изменив его, дописав в секцию установки (do_install_append) замену конфигурационных файлов по умолчанию, файлами, описанными выше: zones, interfaces, policy, masq, rules, routestopped и shorewall.conf

meta-router-bs/recipes-connectivity/shorewall/shorewall_4.4.27.bb:

```
LICENSE = "GPL"
LIC_F = "file://${COREBASE}/meta/files/common-licenses/GPL-1.0-or-later"
SUM_F = "md5=30c0b8a5048cc2f4be5ff15ef0d8cf61"
LIC_FILES_CHKSUM = "${LIC_F};${SUM_F}"

SUB_PR = ".3"
require shorewall.inc
# this version (4.4) requires some deps (perl)
require shorewall-deps.inc

S = "${WORKDIR}/${PN}-${PV}${SUB_PR}"

# запуск стартового скрипта /etc/init.d/shorewall на определенном уровне исполнения
# последовательность запуска, перед сервисом ssh и dhcp
###inherit update-rc.d
###INITSCRIPT_NAME = "shorewall"
###INITSCRIPT_PARAMS = "start 08 2 3 4 5 . stop 92 0 6 1 ."

inherit systemd
SYSTEMD_AUTO_ENABLE = "enable"
SYSTEMD_SERVICE_${PN} = "shorewall.service"

PR = "${INC_PR}.0"
SRC_URI = "\
    http://www.shorewall.net/pub/shorewall/4.4/shorewall-${PV}/${PN}-${PV}${SUB_PR}.tar.bz2 \
    file://zones \
    file://interfaces \
    file://policy \
    file://masq \
    file://rules \
    file://routestopped \
    file://shorewall.conf \
    file://systemd/shorewall.service \
    file://systemd/etc-default-shorewall \
    "


# отключаю секцию do_compile, так как /sbin/shorewall является исполняемым скриптом
# и не требует компиляции
do_compile[noexec] = "1"

SRC_URI[md5sum] = "9f0ef6b547526aa33e34941a211ca602"
SRC_URI[sha256sum] = "1f95a04af2cbdd3449aa6fb26ea1b001e7cccd1ad4ed6d7ed8648247ae5d09bb"


do_install_append () {

    install -m 0644 ${WORKDIR}/zones ${D}${sysconfdir}/shorewall/zones
    install -m 0644 ${WORKDIR}/interfaces ${D}${sysconfdir}/shorewall/interfaces
    install -m 0644 ${WORKDIR}/policy ${D}${sysconfdir}/shorewall/policy
    install -m 0644 ${WORKDIR}/masq ${D}${sysconfdir}/shorewall/masq
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    install -m 0644 ${WORKDIR}/rules ${D}${sysconfdir}/shorewall/rules
    install -m 0644 ${WORKDIR}/routestopped ${D}${sysconfdir}/shorewall/routestopped
    install -m 0644 ${WORKDIR}/shorewall.conf ${D}${sysconfdir}/shorewall/shorewall.conf

    # дополнительно устанавливаю примеры конфигураций брандмауэра
    install -d ${D}${docdir}/shorewall/examples
    cp -R ${S}/Samples/* ${D}${docdir}/shorewall/examples/

    # файлы для запуска сервиса под управлением systemd
    install -d ${D}${systemd_unitdir}/system
    install -d ${D}${sysconfdir}/default
    install -m 0644 ${WORKDIR}/systemd/etc-default-shorewall ${D}${sysconfdir}/default/shorewall
    install -m 0644 ${WORKDIR}/systemd/shorewall.service ${D}${systemd_unitdir}/system
}
```

meta-router-bs/recipes-connectivity/shorewall/shorewall.inc:

```
DESCRIPTION = "Shorewall is a high-level tool for configuring Netfilter."
HOMEPAGE = "http://www.shorewall.net/"
LICENSE = "GPLv2"
SECTION = "network"
PRIORITY = "optional"

INC_PR = "r0"

S = "${WORKDIR}/${PN}-${PV}"

RDEPENDS_${PN} += "iptables"
RRECOMMENDS_${PN} = "kernel-module-ip-tables \
                     kernel-module-ip-conntrack \
                     kernel-module-ipt-conntrack \
                     kernel-module-ipt-multiport \
                     kernel-module-ipt-log \
                     kernel-module-ipt-mac \
                     kernel-module-ipt-mark \
                     kernel-module-ipt-masquerade \
                     kernel-module-ipt-pkttype \
                     kernel-module-ipt-reject \
                     kernel-module-ipt-state \
                     kernel-module-ipt-tos \
                     kernel-module-iptable-filter \
                     kernel-module-iptable-mangle \
                     kernel-module-iptable-nat \
                     "

do_install() {
        export PREFIX=${D}
        ${WORKDIR}/${PN}-${PV}${SUB_PR}/install.sh
}

FILES_${PN} += "/usr/share/shorewall/*"
```

meta-router-bs/recipes-connectivity/shorewall/shorewall-deps.inc:

```
# version 4.4.x requires perl + some perl modules
RDEPENDS_${PN} += "\
    perl \
    perl-module-lib \
    perl-module-autouse \
    perl-module-cwd \
    perl-module-file-basename \
    perl-module-file-temp \
    perl-module-getopt-long \
    perl-module-carp \
    perl-module-findbin \
    perl-module-file-spec-unix \
    perl-module-scalar-util \
    perl-module-io \
    perl-module-io-handle \
    perl-module-exporter-heavy \
    perl-module-list-util \
    perl-module-socket \
    perl-module-overloading \
    perl-module-file-spec \
    perl-module-symbol \
"
```

10. Рецепт сборки образа Router-bs

Для того, чтобы в Yocto Project собрать минималистический дистрибутив необходимо добавить файл рецепта router-bs-image.bb в слой «meta-router-bs»

meta-router-bs/recipes-router-bs/images/router-bs-image.bb:

```
# Минималистический дистрибутив Linux, выполняющий функции "Маршрутизатора"
# для платформы Raspberry PI собранный в "Yocto Project",
# autor Alexander Demachev, site berserk.tv
DESCRIPTION = "The Router BS -  is a simple image to Raspberry PI platform"
LICENSE = "MIT"
MD5_SUM = "md5=0835ade698e0bcf8506ecda2f7b4f302"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;${MD5_SUM}"

IMAGE_FSTYPES_rpi = "rpi-sdimg"
IMAGE_FSTYPES_orange-pi-zero = "sunxi-sdimg"

# добавление нескольких стандартных пакетов в базовый образ
IMAGE_FEATURES += "ssh-server-openssh splash"

# Образ базируется на рецепте сборки содержащем минимальный набор пакетов
# Base this image on core-image-minimal
include recipes-core/images/core-image-minimal.bb

# Установка пароля по умолчанию для пользователя "root"
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
# - основная административная учетная запись в системе
# Set default password for 'root' user
inherit extrausers
ROOTUSERNAME = "root"
ROOTPASSWORD = "routerbs"
EXTRA_USERS_PARAMS = "usermod -P ${ROOTPASSWORD} ${ROOTUSERNAME};"

# стартовая заставка, которая выводиться во время загрузки,
# в случае подключения кабеля HDMI к монитору или к телевизору
SPLASH = "psplash-berserk"

################################################################################
# набор установочных пакетов входящих в образ, разбитый на категории
################################################################################

# отладочные пакеты
# ROUTER_DEBUG_TOOLS = "ldd strace ltrace"

# пакеты библиотеки libc входящие в образ, локализация,
# символьные таблицы для различных языков и т.п.
ROUTER_GLIBC = " \
            glibc-thread-db \
            glibc-gconv-utf-16 \
            glibc-gconv-utf-32 \
            glibc-gconv-cp1251 \
            glibc-binary-localedata-en-us \
            glibc-binary-localedata-ru-ru \
            glibc-charmap-cp1251 \
            glibc-charmap-utf-8 \
            "

# базовые пакеты
ROUTER_BASE = " \
           kernel-modules \
           pciutils \
           "

# сетевые пакеты и модули ядра
ROUTER_NET = "openssh-sftp-server"


# приложения входящие в образ
ROUTER_SOFT = " \
        mc \
        kea \
        init-ifupdown \
        shorewall \
        shorewall-doc \
        "


# указание всех дополнительных пакетов дистрибутива "Router-bs"
# Include modules in rootfs
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
IMAGE_INSTALL += " \
    ${ROUTER_BASE} \
    ${ROUTER_GLIBC} \
    ${ROUTER_NET} \
    ${ROUTER_SOFT} \
    "



#    ${ROUTER_DEBUG_TOOLS} \
#

SHOREWALL_LIST = "\
  shorewall/zones \
  shorewall/interfaces \
  shorewall/policy \
  shorewall/masq \
  shorewall/rules \
  shorewall/routestopped \
  shorewall/shorewall.conf \
"

KEA_LIST = "kea/kea-dhcp4.conf"

ROOTFS_POSTPROCESS_COMMAND += "add_config_git_local;"
# добавление используемых конфигурационных файлов в локальную .git базу
# для удобства отслеживания изменений
add_config_git_local() {
    cd ${IMAGE_ROOTFS}/etc
    git init
    git add ${SHOREWALL_LIST}
    git add ${KEA_LIST}
    git commit -a -m "add shorewall config"
    echo "*" >> .git/.gitignore
}
```

11. Поддерживаемые платформы слоя meta-sunxi и слоя meta-raspberrypi

Ниже приведем общей список поддерживаемых платформ слоя «meta-sunxi», сборка и функционирование дистрибутива сетевого шлюза была проверена только для плат «Orange Pi R1» и «Orange Pi Zero», но возможность выбора достаточно большая.

- bananapi;
- bananapi-m2m;
- bananapi-m2plus;
- bananapi-m64;

– cubieboard2;
– cubieboard;
– cubietruck;
– forfun-q88db;
– licheepi-zero;
– marsboard-a10;
– mele;
– meleg;
– nanopi-m1;
– nanopi-m1-plus;
– nanopi-neo2;
– nanopi-neo-air;
– nanopi-neo;
– nanopi-neo-plus2;
– olinuxino-a10lime;
– olinuxino-a10s;
– olinuxino-a13;
– olinuxino-a13som;
– olinuxino-a20;
– olinuxino-a20lime2;
– olinuxino-a20lime2-emmc;
– olinuxino-a20lime;
– olinuxino-a20som;
– olinuxino-a64;
– orange-pi-one;
– orange-pi-pc;
– orange-pi-pc-plus;
– orange-pi-zero;
– orange-pi-zero-plus2;
– orange-pi-zero-plus2-h3;
– pcduino3;

– pcduino;

– pine64-plus.

Ну и конечно мы не могли обойти своим вниманием самую распространенную платформу в среде «Гиков» - Raspberry Pi. Для сборки дистрибутива под эту плату в «Yocto Project» необходимо использовать слой «meta-raspberrypi». Этот BSP слой поддерживает следующие устройства (для некоторых плат доступна сборка под разрядность 64 бита):

– raspberrypi0;

– raspberrypi0-wifi;

– raspberrypi2;

– raspberrypi3-64;

– raspberrypi3;

– raspberrypi4-64;

– raspberrypi4;

– raspberrypi-cm3;

– raspberrypi-cm;

– raspberrypi.

Примечание. вся функциональность касающаяся платы Raspberry Pi в нашем примере находится на ветке «rpi-gatesgarth», см. пункт ниже (загрузка проекта с github).

12. Краткая инструкция по сборке дистрибутива

1) Установите зависимости для Yocto Project в Ubuntu:

```
sudo apt-get install -y --no-install-suggests --no-install-recommends \
     gawk wget git-core diffstat unzip texinfo gcc-multilib \
     build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
     xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
     xterm
```

2) Установите repo:

```
mkdir ~/bin
curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

3) Загрузите проект с github:

```
PATH=${PATH}:~/bin
mkdir router-bs
cd router-bs
repo init -u https://github.com/berserktv/bs-manifest -m orange/hardknott/router-bs-0.7.4.xml
repo sync
```

Примечание. для платы Raspberry Pi 2/3

repo init -u https://github.com/berserktv/bs-manifest -m raspberry/gatesgarth/router-bs-0.7.3.xml

4) Соберите проект:

```
./shell.sh
bitbake router-bs-image
```

5) Время сборки дистрибутива:

```
Время сборки образа дистрибутива достаточно длительное
и может занимать от пяти до N часов в зависимости от производительности
компьютера, также в процессе сборки из "Интернета" должны быть
загружены исходные коды всех программ входящих в дистрибутив,
часто это полные git базы

(т.е. время сборки также зависит от скорости подключения сети "Интернет")

например на машине: Процессор     - Intel(R) Core(TM)i5-3570 CPU@3.40GHz
                    ОЗУ           - 8 Гбайт
                    Жесткий диск  - внутренний SATA 3Тбайт

                    Время сборки  - от 5 часов и более
                    Размер образа - 194 Мбайт

Размер каталога build после завершения сборки
(cache сборки, исходный код всех программ входящих в образ,
в разных форматах, в случае с git базами они загружается целиком
со всей историей, промежуточные файлы сборки, объектные файлы,
файлы пакетов и т.п.)
занимает примерно 36 ГБайт (du -sh build)

Размер каталога downloads
(git базы приложений и tar.gz архивы)
занимают 3.7 Гбайт
```

5) Запишите дистрибутив на карту памяти - «microSDHC».

13. Административный пароль по умолчанию

В образе Router-bs по умолчанию включена учетная запись:

```
root - пользователь;
routerbs - пароль по умолчанию.
```

ВНИМАНИЕ. Обязательно смените пароль по умолчанию на собственный. Пароль можно сменить выполнив команду passwd на самом устройстве, или в случае самостоятельной сборки в файле рецепта router-bs-image.bb.

14. Первое включение устройства

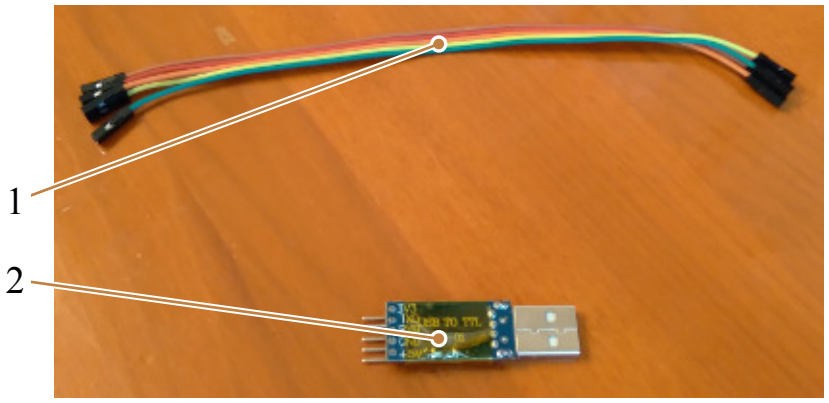
В том случае если вы не перенастроили проект под ваше сетевое окружение, а сразу включили Orange Pi R1, вы можете подключиться к вашему шлюзу по протоколу SSH (команда ssh root@192.168.0.1 с другого компьютера) по ip адресу по умолчанию: 192.168.0.1.

Для этого вы должны подключить кабель к первому сетевому интерфейсу (eth0), если вы не знаете, какой интерфейс у вас первый, то вы можете последовательно подключить сетевой кабель вначале к одному интерфейсу и проверить, ответ на команду ping, а затем и ко второму. Тот интерфейс, который первым ответит и будет искомым. Если вы не получили ответ ни по одному сетевому интерфейсу, то вначале проверьте находитесь ли вы в правильной подсети 192.168.0.0.

Если все равно ответ не получен, то что то пошло не так и в этом случае мы можете подключить технологический последовательный интерфейс UART к Orange Pi R1 и таким образом понять состояние устройства.

15. Отладка платы Orange Pi R1 (UART подключение)

Иногда возникает необходимость понять состояние устройства после обновления прошивки, когда что то не работает. На плате Orange Pi R1 находиться встроенный последовательный UART TTL интерфейс. Для отладки вам необходимо приобрести следующий набор, состоящий из кабеля для подключения (поз.1, рисунок 1) и переходника USB-UART, например PL2303 (поз.2, рисунок 1).



1 – отладочный кабель;
2 – переходник USB-UART (PL2303);

Рисунок 1 - Отладочный набор для подключения к плате R1

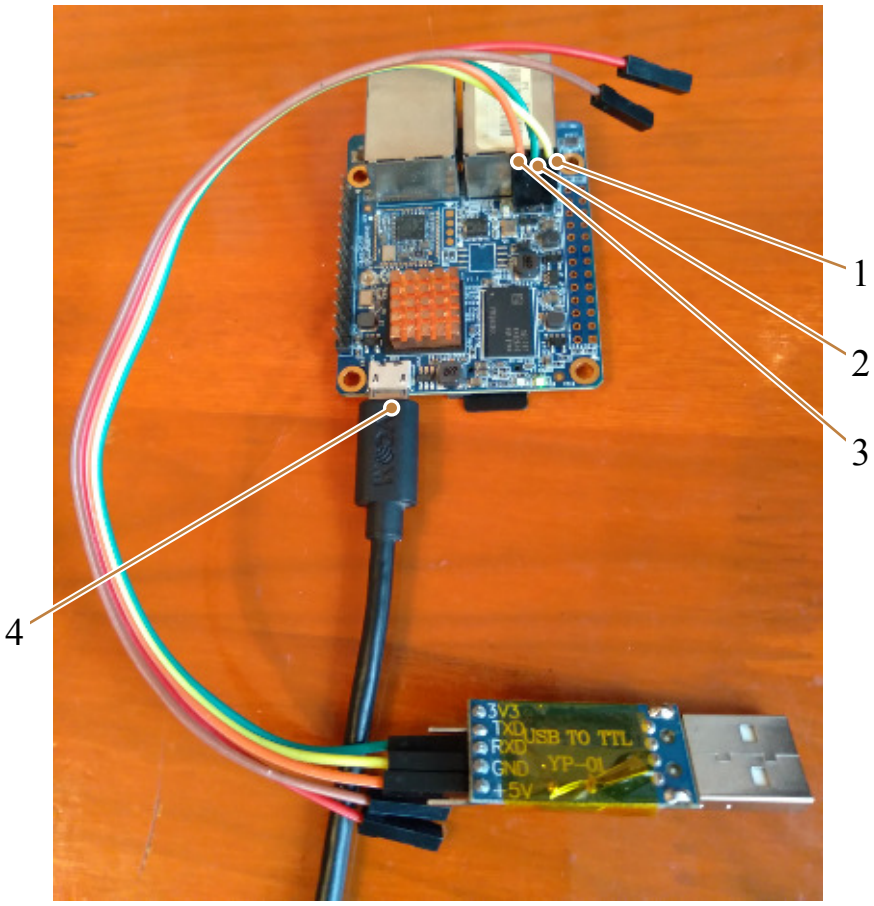
Для отладки вам необходимо выполнить следующие действия:

1) Соедините переходник USB-UART (PL2303) и плату Orange Pi R1 кабелем:

– TX на плате R1 соединяется с контактом RX на переходнике (поз.1, рис.2);

– RX на плате R1 соединяется с контактом TX на переходнике (поз.2, рис.2);

– GND на плате R1 соединяется с контактом GND на переходнике (поз.3, рис.2).



1 – TX на плате Orange R1
2 – RX на плате Orange R1
3 - GND на плате Orange R1
4 - кабель питания платы Orange R1

Рисунок 2 - Подключение платы Orange Pi R1 к компьютеру

2) Подключите переходник USB-UART к компьютеру;

3) Запустите команду в консоле для подключения к последовательному порту:

```
sudo bash
picocom --baud 115200 /dev/ttyUSB0
```

4) Включите адаптер питания платы Orange Pi R1;

5) В консоле появиться вывод штатных команд загрузки и приглашение на ввод имени пользователя. После ввода имени пользователя (root) и пароля (routerbs) вы можете посмотреть логи состояния системы и определить причину неисправности.

16. Запись прошивки на карту памяти microSDHC

Собранный образ будет располагаться по пути:

```
build/tmp/deploy/images/orange-pi-zero
```

На последний успешно собранный образ всегда будет указывать символическая ссылка:

```
router-bs-image-orange-pi-zero.sunxi-sdimg
т.е.
build/tmp/deploy/images/orange-pi-zero/router-bs-image-orange-pi-zero.sunxi-sdimg
```

сам файл будет содержать достаточно длинное имя с датой создания и временем в UTC например такое: router-bs-image-orange-pi-zero-20210914164236.rootfs.sunxi-sdimg.

Для записи прошивки на карту памяти используется команда dd. Запуск команды dd выполняется от имени администратора, при этом необходимо указать входной файл и устройство подключения карты памяти «microSDHC», например /dev/sdX, где X может быть a,b,c и т.д. в зависимости от предыдущих подключенных дисковых разделов.

Проще всего название определить по выводу команды blkid. Наберите команду sudo blkid до подключения карты памяти - «microSDHC» и далее наберите команду sudo blkid после подключения карты памяти - «microSDHC» и тот новый раздел который появился в выводе последней команды и будет искомым. Отмонтируйте разделы карты памяти - «microSDHC» например так, если разделов несколько:

```
umount /dev/sdX1
umount /dev/sdX2
```

В команде dd указывается имя диска целиком т.е. без номера раздела так как в образе содержатся таблицы двух разделов, первый раздел загрузочный диск формата fat16, и второй раздел ext4 - корневая файловая система ОС Linux

ВНИМАНИЕ. все предыдущие данные на карте памяти «microSDHC» после выполнения операции записи будут удалены.

```
sudo bash
cd build/tmp/deploy/images/orange-pi-zero
dd if=router-bs-image-orange-pi-zero.sunxi-sdimg of=/dev/sdX bs=1M
sync
```

17. Некоторые дополнительные функции администрирования

Для упрощения администрирования в проект добавлены несколько bash скриптов со следующими функциями для Linux Desktop клиента:

1. Настройка авторизации на шлюзе с помощью ssh ключей. Генерируется пара из открытого и закрытого ключа. Открытый ключ прописывается на шлюзе, что позволяет не вводить пароль ssh вручную;
2. Подключение удаленного каталога с конфигурацией шлюза (каталог «/etc») в вашу локальную директорию по протоколу sshfs;
3. Редактирование конфигурации в удобном графическом редакторе Visual Studio Code или Kate.

Bash скрипты находятся в каталоге manage:

```
├── COPYING.MIT
├── manage
│   ├── config.txt
│   ├── gen-send-ssh-key.sh
│   ├── kate
│   │   ├── client.sh
│   │   └── install.sh
│   └── vscode
│       ├── client.sh -> ../kate/client.sh
│       └── install.sh
├── meta-router-bs
└── README.md
```

Общие переменные скриптов находятся в sources/router-bs/manage/config.txt:

```
# IP адрес шлюза
IP_GATEWAY="10.0.8.1"

# название ssh ключей, которые будут сохранены в каталоге ~/.ssh
KEY_ID="routerbs_id_rsa"

# временный каталог в который будет монтироваться конфигурация шлюза ("/etc") по протоколу sshfs
CONF_DIR="tmp_etc"
```

Настройка авторизации на шлюзе с помощью ssh ключей:

```
cd sources/router-bs/manage
./gen-send-ssh-key.sh
```

Примечание. вы также можете указать первым аргументом адрес шлюза (по умолчанию он берется из config.txt)

./gen-send-ssh-key.sh 10.0.8.1

Монтирование конфигурации шлюза и редактирование ее в Visual Studio Code:

1. Установка Visual Studio Code выполняется однократно командой:

```
cd sources/router-bs/manage/vscode
./install.sh
```

2. Подключение конфигурации по протоколу sshfs:

```
cd sources/router-bs/manage/vscode
./client.sh
```

Скрипт запускает редактор в фоне и ожидает завершения работы с ним нажатием любой клавиши. После этого, скрипт client.sh выполнит операцию размонтирования. Если по какой то причине монтируемый tmp_etc каталог или файлы в нем используется, например открыт в редакторе или в терминале, то каталог не сможет быть размонтирован, в этом случае необходимо закрыть все программы, которые могут работать с точкой монтирования tmp_etc и заново запустить команду:

```
cd sources/router-bs/manage/vscode
fusermount -u tmp_etc
```

Примечание. Если операция все равно не выполняется, посмотрите есть ли какой нибудь запущенный процесс использующий каталог, командой:

lsof | grep tmp_etc

Если есть остановите этот процесс по его номеру, и снова выполните fusermount.

Монтирование конфигурации шлюза и редактирование ее в редакторе Kate:

```
cd sources/router-bs/manage/kate
# установка редактора Kate (выполняется однократно)
./install.sh
./client.sh
```

И еще, используемая конфигурация Shorewall и файл настройки DHCP сервера Kea добавлены в локальную git базу т.е. /etc/.git на шлюзе. Это может пригодиться для отслеживания изменений конфигурации шлюза, а в редакторе Kate позволяет получить информацию о структуре файлов, которая выводится во вкладке проекты (см. вкладку в левом верхнем углу окна редактора).

Выводы:

Итак в заключении хотелось бы отметить, что «Yocto Project» достаточно мощный инструмент для сборки специализированных дистрибутивов для встраиваемых систем. В нашем руководстве, мы показали как собрать прошивку с крайне ограниченным набором функций для платы Orange Pi R1, ну и для Raspberry Pi 2/3, куда без нее.