

Министерство образования Республики Беларусь

Учреждение образования  
«Гомельский государственный технический  
университет имени П. О. Сухого»

Кафедра «Промышленная электроника»

**А. В. Сахарук**

**ПРОЕКТИРОВАНИЕ УПРАВЛЯЮЩИХ  
И ИНФОРМАЦИОННЫХ СРЕДСТВ  
НА БАЗЕ ЕМБЕДЕД СИСТЕМ**

**ПРАКТИКУМ**

**по выполнению лабораторных работ  
по одноименной дисциплине для студентов  
специальности 1-53 01 07 «Информационные  
технологии и управление в технических системах»  
дневной формы обучения**

**Гомель 2021**

УДК 004.451(075.8)  
ББК 32.972.11я73  
С22

*Рекомендовано научно-методическим советом  
факультета автоматизированных и информационных систем  
ГГТУ им. П. О. Сухого  
(протокол № 10 от 01.06.2020 г.)*

Рецензент: доц. каф. «Информационные технологии» ГГТУ им. П. О. Сухого  
канд. техн. наук, доц. *В. С. Захаренко*

**Сахарук, А. В.**

С22

Проектирование управляющих и информационных средств на базе Ембедед систем : практикум по выполнению лаборатор. работ по одноим. дисциплине для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» днев. формы обучения / А. В. Сахарук. – Гомель : ГГТУ им. П. О. Сухого, 2021. – 92 с. – Систем. требования: РС не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: <https://elib.gstu.by>. – Загл. с титул. экрана.

Содержит пять лабораторных работ с основными теоретическими сведениями, порядком их выполнения, заданиями для самостоятельной работы и контрольными вопросами.

Для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» дневной формы обучения.

**УДК 004.451(075.8)  
ББК 32.972.11я73**

© Учреждение образования «Гомельский  
государственный технический университет  
имени П. О. Сухого», 2021

# Лабораторная работа № 1

## Применение qemu для кросс-компиляции

### 1. Цель работы

Изучить принципы работы с эмулятором qemu, а так же освоить основные приемы использования данного эмулятора для кросс компиляции.

### 2. Основные теоретические сведения

Основной проблемой кросс-компиляции заключается в том, что целевая архитектура отличается от текущей архитектуры. Целевая архитектура — это архитектура процессора, на котором должно работать программное обеспечение, а текущая архитектура — это архитектура процессора, на котором производится разработка и компиляция программного обеспечения.

К методам кросс-компиляции прибегают по разным причинам, но зачастую это связано с тем, что мощность целевого процессора намного меньше и для разработки этого не достаточно.

#### 2.1 Эмулятор qemu и поддерживаемые платформы

Qemu это свободно распространяемый эмулятор, который позволяет эмулировать различные платформы и архитектуры. А так же различные периферийные устройства, такие как:

- Узел i440FX моста PCI и мост PIIX3 PCI к ISA
- Карту Cirrus CLGD 5446 PCI VGA, или поддельную карту VGA с расширением VESA (железный уровень, включающий все нестандартные режимы)
- Мышь и клавиатуру PS/2
- Два интерфейса PCI IDE с поддержкой жёстких дисков и CD-ROM
- Флоппи диск
- Сетевой адаптер NE2000 PCI
- Последовательные порты
- Звуковую карту Creative SoundBlaster 16
- Звуковую карту ENSONIQ AudioPCI ES1370
- Adlib(OPL2) - микросхема совместимая с Yamaha YM3812

- Контроллер PCI UHCI USB и виртуальный USB хаб SMP поддерживает до 255 процессоров. Отметьте, что `adlib` доступен лишь когда QEMU сконфигурирован с `-enable-adlib`. QEMU использует BIOS PC от проекта Bochs, и Plex86/Bochs LGPL VGA BIOS. QEMU использует эмуляцию YM3812 от Tatsuyuki Satoh.

Использование `qemu [options] [disk_image]`

Рассмотрим основные параметры `qemu` в таблице 1.1

Таблица 1.1. Основные параметры `qemu`

Опция	Описание
<code>disk_image</code>	сырой образ жёсткого диска для нулевого жёсткого диска IDE.
<code>-M machine</code>	Выбор эмулируемой машины ("-M ?" для списка (в текущей версии, не выводит - прим. lissyara))
<code>-fda file</code>	Использовать <u>файл</u> как образ гибкого диска 0/1. Вы можете использовать привод гибких дисков на машине, указав как имя файла <code>-/dev/fd0</code> .
<code>-hda file</code> <code>-hdb file[/b</code> <code>-hdc file[/b</code> <code>-hdd file[/b</code>	Использовать <u>файл</u> как образ жёсткого диска 0, 1, 2 или 3.
<code>-cdrom file</code>	Использовать <u>файл</u> как образ CD-ROM (вы не можете использовать <b>-hdci</b> <b>-cdrom</b> одновременно). Вы можете использовать CD-ROM на машине, используя как имя файла <code>/dev/cdrom</code> .
<code>-boot [a c d n]</code>	Загружаться с floppy (a), жёсткого диска (c), CD-ROM (d), или Etherboot (n). По умолчанию - загрузка с жёсткого диска.
<code>-snapshot</code>	Записывать во временные файлы, вместо образов жёстких дисков. В этом случае, используемый вами образ сырого жёсткого диска не записывается. Однако, вы можете вызвать принудительную запись, путём нажатия C-a s.

-no-fd-bootchk	Отключить проверку загрузочной записи гибких дисков в Vochs BIOS. Это может быть необходимо для загрузки со старых гибких дисков.
-m megs	Установить виртуальный размер RAM в <u>megs</u> мегабайт. По умолчанию — 128Mb.
-smp n	Моделировать SMP систему с <u>n</u> CPU. На целевом PC поддерживаться вплоть до 255 CPU.
-nographic	Обычно, QEMU использует SDL для показа вывода VGA. С этой опцией, вы можете полностью отключить графический вывод, таким образом, QEMU - просто приложение командной строки. Эмулируемый последовательный порт - перенаправляется в консоль. Поэтому, вы всё ещё можете использовать QEMU для отлаки ядра Linux с последовательной консолью. (Незаслуженно обижены остальные дистрибутивы гораздо более правильных и прямых ОСей - FreeBSD, например - прим. lissyara)
-vnc display	Обычно, QEMU использует SDL для показа вывода VGA. С этой опцией, вы можете иметь слушающий QEMU на дисплее VNC <u>display</u> , и перенаправлять дисплей VGA через сессию VNC. Это очень полезно для включения устройств usb tablet, при использовании этой опции (опция - <b>usbdevice tablet</b> ). Когда используется дисплей VNC, вы можете использовать опцию <b>-k</b> , для задания раскладки клавиатуры, если вы не используете en-us. <u>Display</u> может быть в форме <u>interface:d</u> , в случае чего, соединения будут разрешены лишь от <u>interface</u> на <u>d</u> . Опционально, <u>interface</u> может быть опущена. <u>display</u> , также, может быть в форме <u>unix:path</u> , где <u>path</u> - расположение сокета, для приёма подключений.

-k language	Использовать раскладку клавиатуры <u>language</u> (например, "fr" для French). Эта опция нужна лишь там, где не безопасно брать сырые коды клавиш PC (например, на Макинтошах, с некоторыми серверами X11, или с дисплеем VNC). Обычно, вам не нужно использовать её на хостах PC/Linux or PC/Windows.
-audio-help	Позволяет показать помощь подсистемы аудио: список драйверов, настраиваемые параметры.
-soundhw card1,card2,... or -soundhw all	Включение звука и выбор аудио оборудования. Используйте ? для вывода всего доступного аудио оборудования.
-localtime	Установка часов реального времени в локальное время (по умолчанию - время UTC). Эта опция необходима для установки корректного времени в MS-DOS или Windows.
-full-screen	Запуститься в полноэкранном режиме.
-pidfile file	Сохранить PID процесса QEMU в <u>file</u> . Это полезно, если вы запускаете QEMU из скрипта.
-daemonize	"Демонизировать" процесс QEMU после инициализации. QEMU не будет отключаться от стандартного IO пока он готов принимать соединения на любых его устройствах. Эта опция - полезна для запуска QEMU внешними программами, без необходимости преодолевать условия запуска.
-option-rom file	Загрузить содержимое файла как опциональную ROM. Эта опция полезна для загрузки вещей, типа EtherBoot.
-usb	Включить драйвер USB (по умолчанию).
-usbdevice devname	Добавить USB устройство <u>devname</u> .

<p><code>-net nic[,vlan=n][,macaddr=addr][,model=type]</code></p>	<p>Создать новую сетевую карту, и подключить её к VLAN`у <u>n</u> (по умолчанию, <u>n</u> = 0). На целевом PC, в настоящее время, NIC - NE2000. Опционально, может быть изменён MAC-адрес. Если опции <b>-net</b> не задано, создаётся одна сетевая карта. QEMU может эмулировать несколько различных моделей сетевых плат. Допустимые значения для <u>type</u> - "ne2k_pci", "ne2k_isa", "rtl8139", "smc91c111" и "lance". Не все устройства поддерживаются на всех целях.</p>
<p><code>-net user[,vlan=n][,hostname=name]</code></p>	<p>Использовать сетевой стек пользовательского режима, не требующий прав администратора для запуска. Для задания имени клиентского хоста, сообщаемого встроенному DHCP-серверу, может использоваться <b>hostname=name</b>.</p>
<p><code>-net tap[,vlan=n][,fd=h][,ifname=name][,script=file]</code></p>	<p>Подключить сетевой интерфейс TAP, хоста, к VLAN`у <u>n</u>, и использовать, для его конфигурирования, скрипт <u>file</u>. По умолчанию, сетевой скрипт - <u>/etc/qemu-ifup</u>. Используйте <b>script=no</b> для отключения выполнения скрипта. Если <u>name</u> не задано, ОС предоставит его автоматически. Для обработки уже открытого хостом TAP-интерфейса, может использоваться <b>fd=h</b>.</p>
<p><code>-net socket[,vlan=n][,fd=h][,listen=[host]:port][,connect=host:port]</code></p>	<p>Подключение VLAN`а <u>n</u> к удалённому VLAN`у, в другой виртуальной машине GEMU, используя соединение через сокет TCP. Если задана <b>listen</b>, QEMU ожидает входящие соединения на <u>port</u> (<u>host</u> - опциональна). <b>connect</b> - используется для подключения к иному экземпляру QEMU, используя опцию <b>listen</b>. <b>fd=h</b> - определяет уже открытый сокет TCP.</p>

<p>-net socket[,vlan=n][,fd=h][,mcast=m addr:port]</p>	<p>Создание VLAN`а <u>n</u>, общего с другой виртуальной машиной QEMU, используя мультикастовый сокет UDP, эффективно делающего шину для каждой QEMU, с тем же самым мультикастовым адресом <u>maddr</u> и <u>port</u>. ПРИМЕЧАНИЕ: 1. Различные QEMU могут быть запущены на различных хостах, и иметь общую шину (предполагается корректная мультикастовая установка для этих хостов). 2. Мультикастовая поддержка - совместима с пользовательским режимом linux 3. Используйте <b>fd=h</b>, для задания уже открытых мультикастовых UDP сокетов.</p>
<p>-net none</p>	<p>Указывает, что сетевые устройства не должны конфигурироваться. Она используется для перезадавания дефолтовой конфигурации (<b>-net nic -net user</b>), которая активируется, если не предоставлена опция <b>-net</b>.</p>
<p>-tftp prefix</p>	<p>Когда используется пользовательский режим сетевого стека, активирует встроенный TFTP-сервер, Все имена файлов, начинающиеся с <u>prefix</u>, могут быть скачаны с хоста на гостя, используя клиент TFTP. Клиент TFTP, на госте, должен быть сконфигурирован в бинарном режиме (используя команду "bin", TFTP клиента UNIX). IP адрес хоста на госте, обычно, - 10.0.2.2.</p>
<p>-smb dir</p>	<p>Когда используется пользовательский режим сетевого стека, активируется встроенный SMB сервер, таким образом, ОС`ы Windows могут иметь прозрачный доступ к файлам хоста в <u>dir</u>.</p>
<p>-kernel bzImage</p>	<p>Использовать <u>bzImage</u> как образ ядра.</p>
<p>-append cmdline</p>	<p>Использовать <u>cmdline</u> как командную строку ядра.</p>
<p>-initrd file</p>	<p>Использовать <u>file</u> как начальный диск в памяти.</p>



-serial dev	<p>Перенаправить виртуальный последовательный порт к хостовому символному устройству <u>dev</u>. Дефолтовое устройство - "vc" в графическом режиме, и "stdio" в неграфическом режиме.</p> <p>Эта опция может использоваться несколько раз, для симуляции вплоть до четырёх последовательных портов.</p>
-parallel dev	<p>перенаправить виртуальный параллельный порт к устройству хоста <u>dev</u> (то же самое устройство, что и последовательный порт). На хостах linux, <u>/dev/parportN</u> может быть использована для использования устройства подключенного на соответствующий параллельный порт хоста.</p>
-monitor dev	<p>перенаправить монитор к устройству хоста <u>dev</u> (то же самое устройство, что и последовательный порт). Устройство по умолчанию - "vc" в графическом режиме, и "stdio" в неграфическом.</p>
-s	<p>ожидать gdb соединения на порт 1234.</p>
-p port	<p>изменить порт соединения gdb. <u>Port</u> может быть одним из десятичных номеров, для задания порта TCP, или устройством хоста (то же самое устройство, что и последовательный порт).</p>
-S	<p>не запускать CPU при запуске (вы можете ввести 'c' в мониторе).</p>
-d	<p>вывести лог в <a href="#">/tmp/qemu.log</a>.</p>
-L path	<p>установить директорию для BIOS, VGA BIOS и раскладки клавиатуры.</p>
-std-vga	<p>моделировать стандартную карту VGA с расширением Voids VBE (по умолчанию - Cirrus Logic GD5446 PCI VGA). Если ваша гостевая ОС поддерживает расширение VESA 2.0 VBE (например, Windows XP), и вы хотите использовать режим высокого разрешения (<math>\geq 1280 \times 1024 \times 16</math>), вы должны использовать эту опцию.</p>

-no-acpi	отключить поддержку ACPI (Advanced Configuration and Power Interface). Используйте её, если ваша гостевая ОС жалуется на проблемы с ACPI (только для целевых машин PC).
-no-reboot	выход, вместо перезагрузки.
-loadvm file	запускаться с сохранённым состоянием ("loadvm" - в монитор).

## 2.2 Изолированная среда

Изолированная среда является своеобразным аналогом виртуальной машины. Однако в отличие от нее при использовании изолированной среды не осуществляется трансляция команд, и все процессы гостевой системы выполняются в том же адресном пространстве что и процессы основной системы. Изолированная среда может быть очень полезна при работе с исходными кодами пакетов, сборке различных программ которые могут привести к падению основной системы. В случае если что то подобное происходит в изолированной среде, то нужно удалить папку с ней и произвести развертывание заново.

Для создания изолированной среды необходимо скачать rootfs нужного дистрибутива и сделать смену корневого каталога для текущего терминала. Далее будет рассматривать пример создания изолированной среды на основе Ubuntu.

Сначала необходимо установить необходимый пакет:

```
sudo apt-get install debootstrap
```

Для развертывания целевого корневого каталога используется одноименная утилита debootstrap. Рассмотрим её основные параметры в таблице 1.2.

Таблица 1.2. Основные параметры утилиты.

Параметр	Описание
--arch=ARCH	Опция задает целевую архитектуру, если она отличается от текущей
-include=alpha,beta	Список дополнительных пакетов, которые необходимо установить. Элементы списка разделяются запятыми
--exclude=alpha,beta	Список пакетов, которые необходимо исключить.
--variant=minbase ...	Имя используемого варианта сценария загрузки. В

	<p>настоящее время поддерживаются варианты:</p> <ul style="list-style-type: none"> <li>☞ minbase, которые включают только необходимые пакеты и apt;</li> <li>☞ buildd, который устанавливает сборку необходимых пакетов в TARGET;</li> <li>☞ fakechroot, который устанавливает пакеты без привилегий root.</li> <li>☞ scratchbox, который предназначен для создания целей для использования с нуля.</li> </ul> <p>По умолчанию, без аргумента --variant = X, устанавливается базовая ерсия Dbeian.</p>
--no-check-gpg	Отключает проверку подписи gpg.
--verbose	Получите больше информации о загрузке.
--download-only	Загрузить пакеты без установки.
--no-check-certificate	Не проверять сертификаты.
--foreign	Выполняйте только начальную фазу начальной загрузки, например, если целевая архитектура не соответствует хосту архитектура.

Использование:

```
sudo debootstrap --arch=i386 --include=gcc,g++,git,nano artful /opt/chroot
ftp://ftp.byfly.by/ubuntu
```

В данном варианте развертывается целевой корневой каталог системы в папке /opt/chroot. Который предназначен для архитектуры i386. Вместе с основными пакетами так же будут установлены gcc,g++ и git. Версия дистрибутива Ubuntu 17.10 (Artful Aardvark). Зеркало репозитория ftp://ftp.byfly.by/ubuntu.

Таблица 1.3. Версии Ubuntu и их кодовые имена

Версия	Кодовое имя	Окончание срока поддержки
12.04 LTS	Precise Pangolin	апрель 2017 года
12.10	Quantal Quetzal	16 мая 2014 года
13.04	Raring Ringtail	27 января 2014 года
13.10	Saucy Salamander	17 июля 2014 года
14.04 LTS	Trusty Tahr	апрель 2019 года
14.10	Utopic Unicorn	23 июля 2015 года
15.04	Vivid Vervet	4 февраля 2016 года
15.10	Wily Werewolf	июль 2016 года

16.04 LTS	Xenial Xerus	апрель 2021 года
16.10	Yakkety Yak	июль 2017 года
17.04	Zesty Zapus	январь 2018 года
17.10	Artful Aardvark	июль 2018 года
18.04	Bionic Beaver	Апрель 2023
18.10	Cosmic Cuttlefish	Июль 2019

После окончания срока поддержки версии Ubuntu удаляются из официального репозитория и основных зеркал и перемещаются в <http://archive.ubuntu.com/ubuntu/>.

Поддерживаемые архитектуры (официальный репозиторий):

- ⑩ i386 (x86 intel)
- ⑩ x86\_64 (amd64)
- ⑩ arm
- ⑩ armhf (hard float)
- ⑩ ppc64el(OpenPOWER)
- ⑩ S390X (IBM Mainframe Systems)

Таблица 1.4. Зеркала репозитория Ubuntu

Ссылка	Расположение
<a href="http://ftp.byfly.by/ubuntu/">http://ftp.byfly.by/ubuntu/</a>	Беларусь
<a href="http://mirror.datacenter.by/ubuntu/">http://mirror.datacenter.by/ubuntu/</a>	
<a href="http://mirror.yandex.ru/ubuntu/">http://mirror.yandex.ru/ubuntu/</a>	Россия
<a href="http://mirror.truenetwork.ru/ubuntu/">http://mirror.truenetwork.ru/ubuntu/</a>	
<a href="http://mirror.corbina.net/ubuntu/">http://mirror.corbina.net/ubuntu/</a>	
<a href="http://ubuntu.volia.net/ubuntu-archive/">http://ubuntu.volia.net/ubuntu-archive/</a>	Украина
<a href="http://ubuntu.ip-connect.vn.ua/">http://ubuntu.ip-connect.vn.ua/</a>	
<a href="http://ubuntu.mirrors.omnilance.com/ubuntu/">http://ubuntu.mirrors.omnilance.com/ubuntu/</a>	
<a href="http://mirror.vpsnet.com/ubuntu/">http://mirror.vpsnet.com/ubuntu/</a>	Литва
<a href="http://ubuntu-archive.mirror.serveriai.lt/">http://ubuntu-archive.mirror.serveriai.lt/</a>	
<a href="http://ubuntu.mirror.vu.lt/ubuntu/">http://ubuntu.mirror.vu.lt/ubuntu/</a>	
<a href="http://ftp.icm.edu.pl/pub/Linux/ubuntu/">http://ftp.icm.edu.pl/pub/Linux/ubuntu/</a>	Польша
<a href="http://mirror.onet.pl/pub/mirrors/ubuntu/">http://mirror.onet.pl/pub/mirrors/ubuntu/</a>	
<a href="http://ftp.agh.edu.pl/ubuntu/">http://ftp.agh.edu.pl/ubuntu/</a>	

Но не все репозитории поддерживают полный список архитектур. Информация по каждому репозиторию можно посмотреть на сайте <https://launchpad.net/ubuntu/+archivemirrors>.

Если не указывается явным образом ссылка на репозиторий, то скачивание происходит из основного репозитория Ubuntu.

### 2.3 Пакет `qemu-static`

Помимо полной эмуляции заданной платформы и периферийных устройств, Qemu имеет пакет `qemu-user-static`. Данный пакет позволяет запускать код, собранный под целевую платформу не создавая полноценной виртуальной машины. При решении общих вопросов кросскомпиляции, не требующих взаимодействия с различными периферийными устройствами, и развертывания различных бинарных пакетов данный режим является приоритетным и менее затратным.

Для установки данного пакета в ОС Ubuntu необходимо выполнить команду:

```
sudo apt install qemu-user-static
```

После удачной установки в системе появятся следующие программы:

- `qemu-aarch64-static`
- `qemu-i386-static`
- `qemu-mipsn32el-static`
- `qemu-ppc64-static`
- `qemu-sparc-static`
- `qemu-alpha-static`
- `qemu-m68k-static`
- `qemu-mipsn32-static`
- `qemu-ppc-static`
- `qemu-tilegx-static`
- `qemu-armeb-static`
- `qemu-microblazeel-static`
- `qemu-mips-static`
- `qemu-s390x-static`
- `qemu-x86_64-static`
- `qemu-arm-static`
- `qemu-microblaze-static`
- `qemu-nios2-static`
- `qemu-sh4eb-static`

- qemu-cris-static
- qemu-mips64el-static
- qemu-or1k-static
- qemu-sh4-static
- qemu-mips64-static
- qemu-ppc64abi32-static
- qemu-sparc32plus-static
- qemu-hppa-static
- qemu-mipsel-static
- qemu-ppc64le-static
- qemu-sparc64-static

Каждая из этих программ рассчитана для запуска приложений под ту или иную архитектуру.

Таблица 1.5 Основные параметры qemu-arm-static

Параметр	Описание
-h -help	Вызов справки
-g <port>	Ожидать подключения отладчика GDB на заданном порте
-L <path>	Установить путь к elf в системную переменную PATH
-s <size>	Задать размер стека
-cpu <model>	Задать модель процессора
-E var=value	Установить значение заданной переменной окружения
-U <var>	Очистить заданную переменную окружения
-r <uname>	Задать строку uname
-B <address>	Установить адрес guest_base
-R <size>	Установить размер гостевой виртуальной памяти
-d <item[,...]>	Включить запись указанных элементов
-dfilter <range[,...]>	Фильтр для журнала по диапазону адресов
-D <logfile>	Задать файл лога
-p <pagesize>	Установите размер страницы хоста
-singlestep	Работать в режиме одиночного шага (singlestep)
-strace	Включить журнал системных вызовов
-version	Вывести версию программы

Таблица 1.6 Поддерживаемые процессоры qemu-arm-static

arm1026	arm1136	arm1136-r2	arm1176	arm11mpcore
arm926	arm946	cortex-a15	cortex-a7	cortex-a8
cortex-a9	cortex-m3	cortex-m4	cortex-r5	pxa250
pxa255	pxa260	pxa261	pxa262	pxa270-a0
pxa270-a1	pxa270	pxa270-b0	pxa270-b1	pxa270-c0
pxa270-c5	sa1100	sa1110	ti925t	any

Среди списка процессоров присутствует такой параметр как any, его следует выбирать в случае если заданный процессор отсутствует в списке. И в данном случае будет производиться эмуляция некоего абстрактного процессора с архитектурой ARM.

### 3. Порядок выполнения работы

Использовать эмулятор qemu можно двумя способами: полная эмуляция, трансляция команд. Для использования первого варианта необходимо создать полный образ целевой системы, а так же настроить эмуляцию аппаратного обеспечения. А для использования второго метода достаточно развернуть изолированную среду под заданную архитектуру.

#### 3.1 Развертывание изолированной среды

Для развертывания изолированной среды необходимо выполнить ряд действий. В общем случае все сводится к следующей последовательности действий

- установка debootstrap
- создание каталога для развертывания среды
- запуск debootstrap с заданными параметрами среды
- скачивание из репозитория необходимых пакетов
- распаковка пакетов
- установка пакетов (при использовании несовместимой целевой архитектуры необходимо запускать установку через chroot)

Рассмотрим пример развертывания изолированной среды. Для этого необходимо выполнить ряд действий.

1. Создаем директорию в которой будет развернута изолированная среда:

```
mkdir /opt/chroot
```

2. Разворачиваем изолированную среду:

```
sudo debootstrap --arch=i386 --include=gcc,g++,git,nano artful /opt/chroot  
ftp://ftp.byfly.by/ubuntu
```

Данную команду можно выполнять с правами суперпользователя (root). Таким образом происходит скачивание и развертывание системы Ubuntu 17.10 под архитектуру intel x86. Так же устанавливаются дополнительные пакеты gcc, g++, git и nano. Скачивание происходит зеркала ftp://ftp.byfly.by/ubuntu. При успешном выполнении операции в указанной папке появится корневая файловая система.

3. Монтируем специальные системные файловые системы:

```
sudo mount --bind /proc /opt/chroot/proc  
sudo mount --bind /sys /opt/chroot/sys  
sudo mount --bind /dev /opt/chroot/dev
```

Данная процедура необходима для того чтобы изолированная среда имела доступ к системным устройствам, доступ в интернет для установки пакетов через apt и т.д....

Для того чтобы каждый раз после перезагрузки системы не приходилось производить монтирование заново, можно добавить соответствующие запись в /etc/fstab

4. Производим смену rootfs для текущей сессии терминала:

```
sudo chroot /opt/chroot
```

При успешном выполнении команды в терминале появится строка приглашения от пользователя root в гостевой системе.

Далее можно приступать к установке необходимых программных пакетов и утилит.

### 3.2 Применение пакета qemu-static

Во данном примере приведен сокращенный способ создания среды для кросскомпиляции на основе QMEU, который по своей сути является аналогом изолированной среды но под целевую платформу.

Для этого мы должны создать папку где будет содержаться изолированная среда, Затем с помощью утилиты debootstrap произвести развертывание целевой корневой файловой системы и выполнить chroot с использованием qemu-static.

Теперь подробнее:



1. Создаем целевой каталог, в котором будет развернута изолированная среда под целевую платформу armhf

```
mkdir /opt/arm
```

2. Зададим переменную среды с версией необходимого дистрибутива Ubuntu

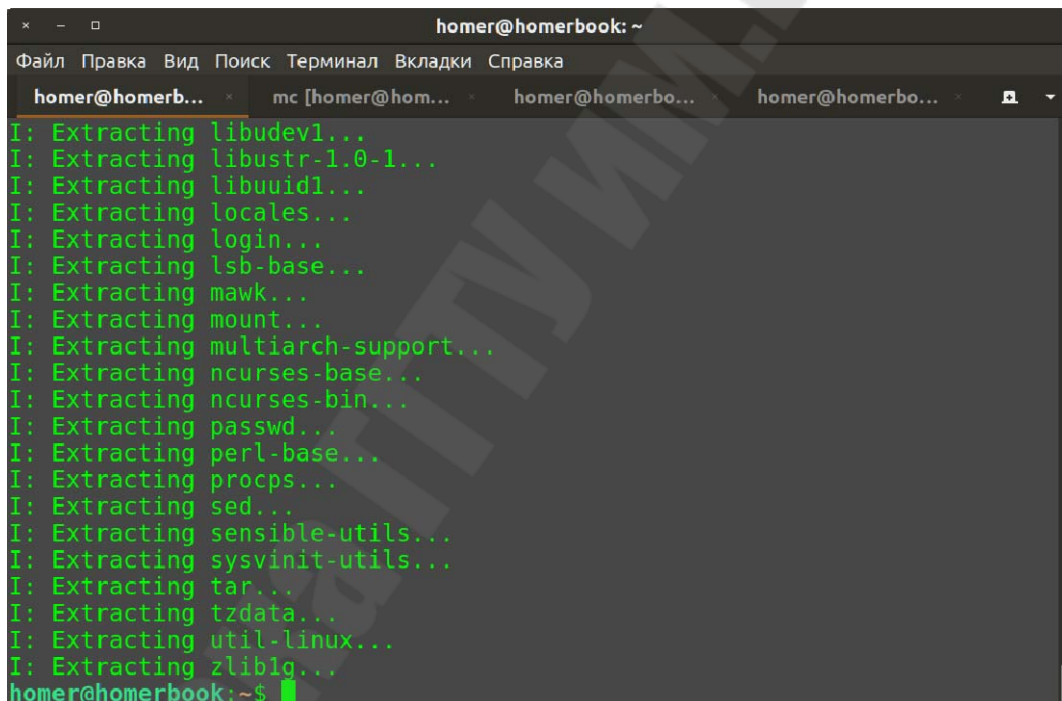
```
export distro=yakkety
```

3. Зададим переменную среды, которая будет содержать путь к целевому каталогу

```
export targetRoot=/opt/arm
```

4. Развертываем целевую корневую файловую систему. Результат приведен на рисунке 1.1

```
sudo debootstrap --arch=armhf --foreign $distro $targetRoot
```



```
homer@homerbook: ~
Файл Правка Вид Поиск Терминал Вкладки Справка
homer@homerb... mc [homer@hom... homer@homerbo... homer@homerbo...
I: Extracting libudev1...
I: Extracting libustr-1.0-1...
I: Extracting libuuid1...
I: Extracting locales...
I: Extracting login...
I: Extracting lsb-base...
I: Extracting mawk...
I: Extracting mount...
I: Extracting multiarch-support...
I: Extracting ncurses-base...
I: Extracting ncurses-bin...
I: Extracting passwd...
I: Extracting perl-base...
I: Extracting procps...
I: Extracting sed...
I: Extracting sensible-utils...
I: Extracting sysvinit-utils...
I: Extracting tar...
I: Extracting tzdata...
I: Extracting util-linux...
I: Extracting zlib1g...
homer@homerbook:~$
```

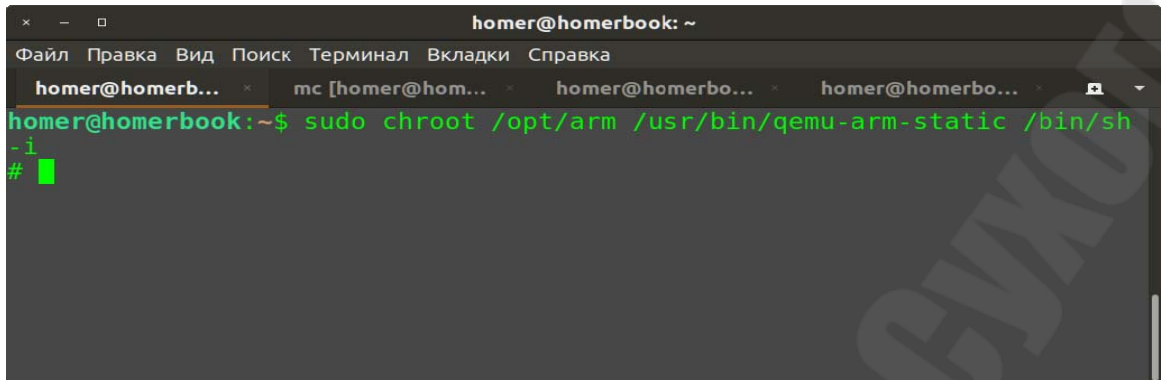
Рисунок 1.1 — Развертывание виртуальной среды

5. Копируем qemu-arm-static в целевой каталог

```
sudo cp /usr/bin/qemu-arm-static $targetRoot/usr/bin/
```

6. Выполняем chroot с использованием qemu. Результат приведен на рисунке 1.2

```
sudo chroot $targetRoot /usr/bin/qemu-arm-static /bin/sh -i
```



```
homer@homerbook: ~  
Файл Правка Вид Поиск Терминал Вкладки Справка  
homer@homerb... x mc [homer@hom... x homer@homerbo... x homer@homerbo... x  
homer@homerbook:~$ sudo chroot /opt/arm /usr/bin/qemu-arm-static /bin/sh  
-i  
# █
```

Рисунок 1.2 – Выполнение chroot в изолированную среду

7. Т.к. изолированная среда разворачивалась для несовместимой платформы то ее установка производится в два этапа. Вторым этапом

```
/debootstrap/debootstrap —second-stage
```

8. Теперь необходимо добавить репозитории в изолированную среду:

```
cat <<EOT > /etc/apt/sources.list  
deb http://ports.ubuntu.com/ $distro main universe  
deb-src http://ports.ubuntu.com/ $distro main universe  
deb http://ports.ubuntu.com/ $distro-security main universe  
deb-src http://ports.ubuntu.com/ $distro-security main universe  
deb http://ports.ubuntu.com/ $distro-updates main universe  
deb-src http://ports.ubuntu.com/ $distro-updates main universe  
EOT
```

9. Выходим из chroot нажав комбинацию клавиш ctrl+d

10. Для того чтобы изолированная среда имела доступ в интернет копируем resolv.conf

```
sudo cp /etc/resolv.conf $targetRoot/etc
```

11. Выполняем полноценный chroot для созданной изолированной среды

```
sudo chroot $targetRoot
```

12. Теперь необходимо установить пакеты для кросскомпиляции

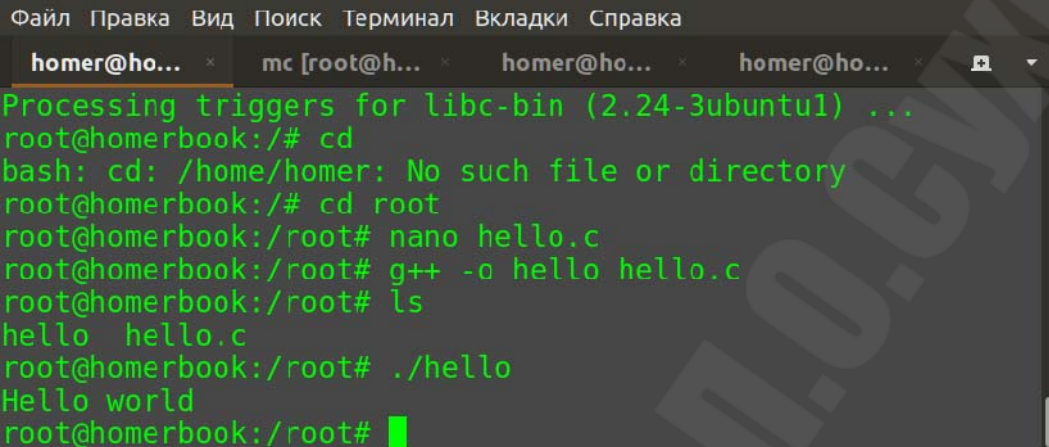
```
apt update  
apt install gcc g++ nano mc
```

13. Для проверки процесса компиляции нужно написать простейшую программу и собрать ее под целевую платформу

```
#include <stdio.h>  
  
int main ()  
{  
    puts ("Hello world");  
}
```

```
    return 0;
}
```

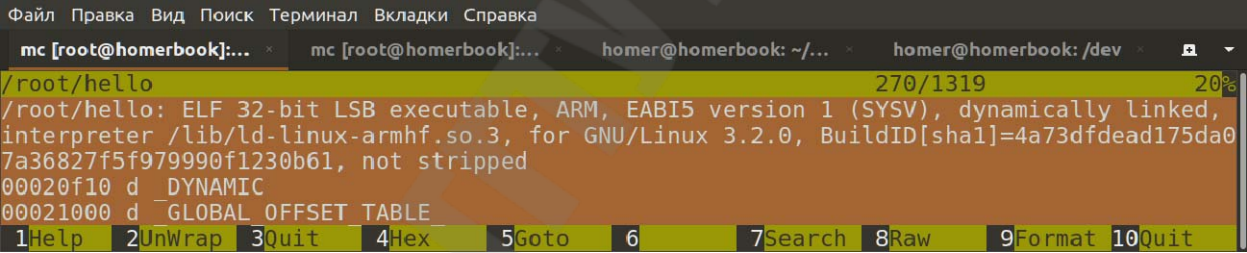
Результат приведен на рисунке 1.3



```
Файл Правка Вид Поиск Терминал Вкладки Справка
homer@ho... x mc [root@h... x homer@ho... x homer@ho...
Processing triggers for libc-bin (2.24-3ubuntu1) ...
root@homerbook:/# cd
bash: cd: /home/homer: No such file or directory
root@homerbook:/# cd root
root@homerbook:/root# nano hello.c
root@homerbook:/root# g++ -o hello hello.c
root@homerbook:/root# ls
hello hello.c
root@homerbook:/root# ./hello
Hello world
root@homerbook:/root#
```

Рисунок 1.3 – Компиляция и запуск приложения в изолированной среде

Можно посмотреть, под какую архитектуру собран исполняемый файл (данную возможность можно найти в mc, открыв исполняемый файл для просмотра)



```
Файл Правка Вид Поиск Терминал Вкладки Справка
mc [root@homerbook]:... x mc [root@homerbook]:... x homer@homerbook: ~/... x homer@homerbook: /dev x
/root/hello 270/1319 20%
/root/hello: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, BuildID[sha1]=4a73dfdead175da0
7a36827f5f979990f1230b61, not stripped
00020f10 d _DYNAMIC
00021000 d _GLOBAL_OFFSET_TABLE
1Help 2UnWrap 3Quit 4Hex 5Goto 6 7Search 8Raw 9Format 10Quit
```

Рисунок 1.4 — Информация о скомпилированном бинарном файле

Как видно из выведенной информации данный исполняемый файл имеет формат ELF под 32-х разрядный процессор архитектуры ARM.

### 3.3 Установка и настройка эмулятора под заданную платформу с полной эмуляцией аппаратного обеспечения

Теперь рассмотрим использование QEMU для виртуализации другой машины в типичной среде настольного ПК под GNU/Linux. Эмуляция другой машины похожа на начало работы с только что купленным новым компьютером. Первый шаг - это установка операционной системы. Новый компьютер, конечно, должен иметь место для установки операционной системы, поэтому необходим жесткий диск.

QEMU предоставляет специальную команду для создания жесткого диска, которая называется `qemu-img`. Эта утилита создает образы различных форматов, но лучший (для `qemu`) из них является `qcow` (или `qemu sору-on-write`). Преимуществом данного формата является то, что образ эмулируемого

диска не обязательно должен занимать физический файл такого же объема. Другими словами, формат допускает пропуски, что позволяет сделать образ диска более компактным. Например, пустой образ диска объемом 4 ГБ займет всего 16 КБ.

Для `qemu-img` необходимо указать операцию (`create` для создания нового образа диска), формат (`qcow` для форматирования образа `qemu`), размер и имя образа диска. Следующий пример эмулирует машину для небольшого дистрибутива Linux, предполагаемого для использования на Flash. Итак, создаем образ диска на 128 МБ:

```
$ qemu-img create -f qcow disk.img 128M
Formating 'disk.img', fmt=qcow, size=131072 kB
```

Теперь, когда жесткий диск создан, можно установить на него новую операционную систему. Для демонстрации этого процесса будет использован небольшой дистрибутив Linux, называемый `cfLinux`, предназначенный для применения в качестве небольшой встраиваемой Linux-системы в таких устройствах, как шлюзы, беспроводные точки доступа, брандмауэры и маршрутизаторы. Этот дистрибутив можно загрузить в формате ISO с помощью `wget`:

```
wget ftp://ftp.cflinux.fu/pub/cflinux/iso/cflinux-1.0.iso
```

Образ ISO представляют собой широко применяемый формат CD-ROM (также известный как файловая система ISO 9660).

Теперь у нас есть эмулируемый диск (`disk.img`) и CD-ROM, с которого можно установить операционную систему. Следующим шагом будет установка системы на жесткий диск. Это делается очень просто с помощью `qemu`:

```
qemu -hda disk.img -cdrom /root/cflinux-1.0.iso -boot d
```

При использовании `qemu` образ жесткого диска задается с помощью опции `hda`, а компакт-диск (файл, где располагается образ) - с помощью опции `cdrom`. Опция `boot` позволяет загрузиться с CD-ROM. Аргумент `d` указывает загрузиться с CD-ROM, а `a` - с флоппи-диска, с `n` указывает на загрузку с жесткого диска (по умолчанию), а `p` - загрузку с сети. Если команда введена правильно, то появится новое окно QEMU с эмулируемой машиной.

Следуя инструкциям по установке с CD-ROM, легко закончить установку с ISO-образа на эмулируемый жесткий диск. Установка требует перезагрузки. В этом месте можно закончить эмуляцию (`Ctrl-C` в окне `qemu`). Теперь можно загрузить свежее установленную операционную систему с помощью следующей команды:

```
qemu -hda disk.img
```

```
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 8192 bind 16384)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
ds: no socket drivers loaded!
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 7997k freed
cramps: wrong magic
UFS: Mounted root (minix filesystem) readonly.
Freeing unused kernel memory: 80k freed
.....

*** Welcome, this is the cflinux installation program. ***

It will ask you where to install cflinux, and will destroy all the contents
of that device.

The following block devices were detected on your system, sorted by size:
  device  size (MB)
  hda     128
On which would you like to install CFLinux? [hda] _
```

Рисунок 1.5 Подготовка к установке cfLinux на эмулируемый диск в QEMU

Эта команда просто эмулирует стандартный PC (опция по умолчанию) с жестким диском, представленным файлом образа disk.img. Образ Linux, загрузившись с эмулируемого жесткого диска, выдаст в результате окно QEMU, показанное на рисунке 1.6

```
insmod: init_module: 8139cp: No such device
modprobe: failed to load module 8139cp
8139too Fast Ethernet driver 0.9.26
insmod: init_module: 8139too: No such device
modprobe: failed to load module 8139too
natsemi dp8381x driver, version 1.07-LK1.0.17, Sep 27, 2002
  originally by Donald Becker <becker@scyld.com>
  http://www.scyld.com/network/natsemi.html
  2.4.x kernel port by Jeff Garzik, Tjeerd Mulder
insmod: init_module: natsemi: No such device
modprobe: failed to load module natsemi
Adding system group nogroup: done
Adding system user nobody: done
Network initialization: done
Enabling TCP/IP SYN cookies: done.
Enabling spoof protection on all interfaces: done.
Setting default ARP announce method(1) on all interfaces: done.
Disabling console blanking: done
Starting standard daemons: syslogd klogd crond [generating 2048 bits rsa key] [g
enerating 1024 bits dsa key] sshd.
Local initialization: .
(none) login: _
```

Рисунок 1.6 - Загрузка свежестановленного cfLinux с эмулируемого жесткого диска

Данный способ весьма объемный, но позволяет производить эмуляцию не только платформы, но и набора периферии. Однако далеко не всегда для задач кросскомпиляции бывает необходима полная эмуляция устройства.

### **3.4 Задание для самостоятельной работы**

Реализовать эмуляцию и скомпилировать простейшее приложение на языке программирования C. Платформу и тип эмуляции (полная или изолированная среда) задает преподаватель.

### **4. Содержание отчета**

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а так же снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Так же обязательно выполнить примеры из данной работы и вставить снимки экрана процесса выполнения примеров. Вывод о полученных навыках и знаний после выполнения работы.

#### **Контрольные вопросы для самопроверки**

1. Что такое кросскомпиляция?
2. Что такое qemu?
3. Как применяется qemu для кросскомпиляции?
4. Какие два основных метода эмуляции заданной платформы существуют?

## Лабораторная работа № 2

### Сборка кросскомпилятора с помощью crosstool-ng

#### 1. Цель работы

Изучить основные принципы сборки кросскомпиляторов с помощью пакета crosstool-ng.

#### 2. Основные теоретические сведения

Сборка кросскомпилятора — это очень важный и ответственный момент, т. к. от него в последствии будет зависеть не только работа разработанного вами программного обеспечения, но и самой операционной системы. В данной работе кросскомпилятор будет собираться на основе пакета GCC. В принципе осуществить сборку кросскомпилятора можно и без использования пакета crosstool-ng, однако это займет больше времени и потребует больше трудозатрат. Пакет crosstool-ng упрощает процесс настройки и сборки кросскомпилятора, а так же имеет утилиту с псевдографическим интерфейсом для удобства конфигурации сборки.

##### 2.1 Получение исходного кода crosstool-NG

Crosstool-NG — это свободно распространяемая, кроссплатформенная конфигурационная система для тулчейна на основе компилятора GCC под заданную платформу. Другими словами это консольная утилита, которая распространяется в виде исходного кода и позволяет сконфигурировать и собрать кросс компилятор и сопутствующие библиотеки и утилиты под заданную архитектуру.

Получить исходный код данного пакета абсолютно бесплатно можно двумя способами:

1. Скачать последнюю версию с официального сайта проекта  
<https://crosstool-ng.github.io>
2. Получить последнюю версию через систему контроля версий GIT с сервера GitHub  

```
git clone https://github.com/crosstool-ng/crosstool-ng
```

Предпочтительнее применять второй способ, т.к. на сервере GitHub чаще выкладываются последние версии пакета с исправленными ошибками и доработками.

В обоих случаях структура исходных кодов будет приблизительно одинаковая, могут быть небольшие отличия в зависимости от версии пакета. Корневая папка исходного кода crosstool-ng имеет вид представленный на рисунке 2.1.

```
homer@homer-book: /opt/tempGCC/crosstool-ng$ ls
bash-completion  contrib  docs  licenses.d  packages  scripts
bootstrap        COPYING  issue_template.md  m4  paths.sh.in  testing
config           ct-ng.in  kconfig  maintainer  README.md  TODO
configure.ac     debian   LICENSE  Makefile.am  samples
homer@homer-book: /opt/tempGCC/crosstool-ng$
```

Рисунок 2.1 — Структура корневой папки исходного кода crosstool-ng.

Более подробно стоит остановиться на папке `samples`. Она содержит примеры конфигурации пакета под различные архитектуры и платформы.

Так же существует официальная документация по сборке и использованию данного пакета. Ее можно просмотреть по ссылке на официальном сайте проекта <http://crosstool-ng.github.io/docs/>.

## 2.2 Сборка crosstool-NG

Пакет crosstool-NG распространяется в виде исходных кодов. Данный факт с одной стороны усложняет его использование, а с другой стороны делает его более универсальным. К минусам данного подхода можно отнести необходимость сборки и установки самого пакета перед использованием, что в свою очередь требует наличия навыков сборки пакетов в среде Linux из исходных кодов, а так же требует затрат по времени на данную процедуру. К плюсам можно отнести универсальность пакета, а именно данный пакет может быть собран под различные архитектуры и соответственно использоваться на них. Например, к таким моментам можно отнести возможность сборки самого пакета на устройстве с архитектурой ARM и последующей сборкой тулчейна для кросскомпиляции на платформе с данной архитектурой пакетов под x86 совместимую архитектуру.

В общем случае сборка пакета crosstool-NG и последующая сборка тулчейна состоит из следующих этапов:

- Получение исходного кода проекта
- Установка необходимых зависимостей для сборки crosstool-NG
- Конфигурация пакета
- Сборка пакетам
- Установка пакета
- Запуск утилиты конфигурации сборки тулчейна
- Запуск сборки тулчейна
- Автоматическое скачивание всех необходимых исходных кодов пакетов для сборки тулчейна
- Сборка всех необходимых пакетов

Установка собранного тулчейна в целевую папку



Библиотека ГГТУ им. П.О.Сухого

## 2.3 Основные параметры для сборки кросс компилятора под целевую платформу

Интерфейс программы для настройки crosstool-ng представляет собой построенное из символов псевдографики окно. Данный метод построения интерфейса применяется из соображений максимальной совместимости и минимальной нагрузки на железо. С помощью такого интерфейса может производиться конфигурация через терминал, даже на системах, не имеющих полноценного графического интерфейса, а так же удаленно с помощью подключения по ssh. Внешний вид основного окна представлен на рисунке 2.2.

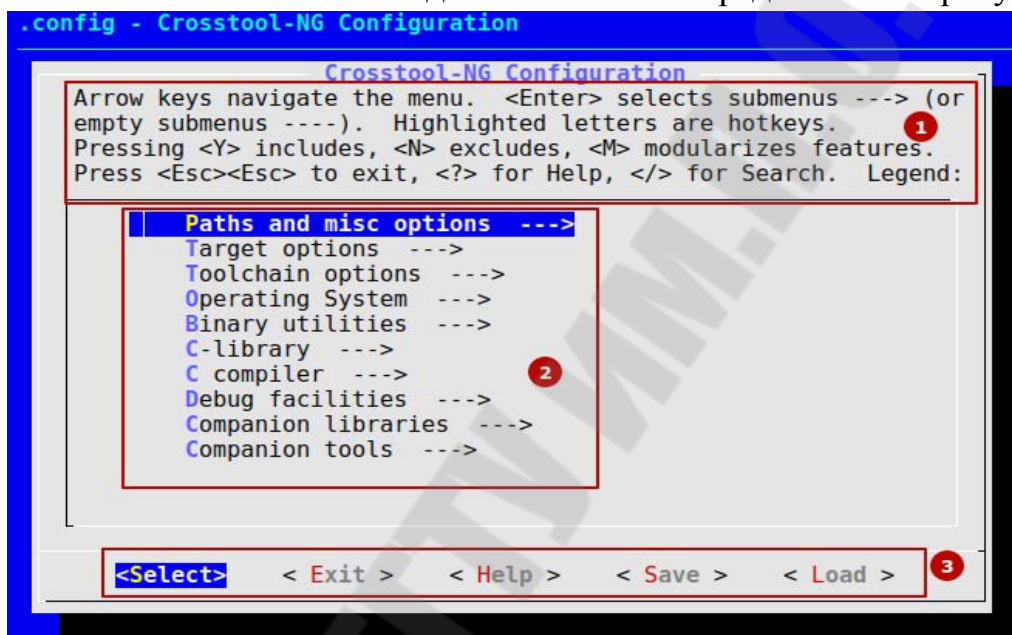


Рисунок 2.2 — Внешний вид основного окна

Данный интерфейс состоит из 3 основных областей:

1. область с краткой инструкцией пользования программой;
2. область основных настроек;
3. область управления.

Для выбора конкретного параметра требуется нажать клавишу <Enter>(или пункт <Select> в области управления). Заглавные буквы настроек являются горячими клавишами для быстрого перехода. Нажатие клавиши <Y> инициализирует включение выбранного пункта в сборку кросс-компилятора. Соответственно клавиша <N> исключает выбранный пункт.

Для выхода из программы следует нажать два раза клавишу <Esc> (с последующим выбором сохранения изменений) (или пункт <Exit> в области управления), одно нажатие <Esc> выводит на уровень выше настроек. Чтобы вызвать окно дополнительной информации требуется нажать клавишу <?> (или пункт <Help> в области управления). Для поиска конфигурационных параметров можно использовать клавишу </>. Выбор пункта <Save> в области управления инициализирует сохранение настроек. Для загрузки

настроек сохраненных в отдельном файле предназначен пункт <Load> с последующим вводом имени файла.

Область основных настроек главного окна содержит следующие пункты:

- Paths and misc options(настройка путей и других опций);
- Target options (опции целевой архитектуры);
- Toolchain options (настройки инструментария);
- Operating System (настройка целевой операционной системы);
- Binary utilities (настройка бинарных утилит);
- C-library (конфигурация библиотек языка C);
- C compiler (настройка компиляции языка C);
- Debug facilities (подключение отладочных средств);
- Companion libraries (подключение сопутствующих библиотек);
- Companion tools (подключение сопутствующих инструментов).

Вкладка Paths and misc options (настройка путей и других опций) представлена на рисунке 2.3.

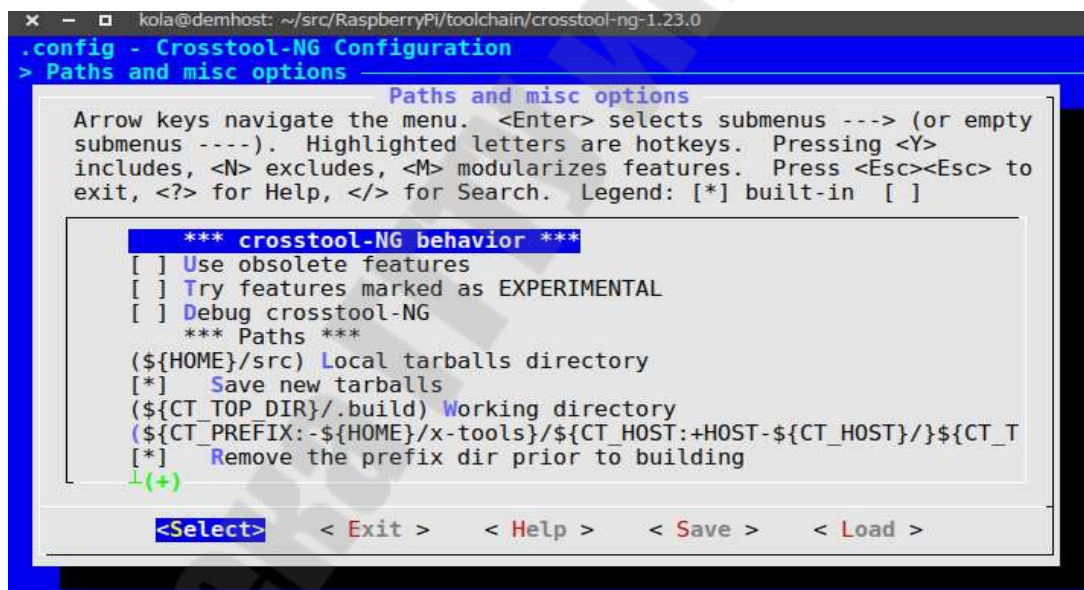


Рисунок 2.3 — Вкладка Paths and misc options

- Use obsolete features (Использовать устаревшие функции).

Данный пункт позволяет включить\исключить использование старых функций для поддержки старых и редко используемых в настоящее время заголовков ядра, старых версий gcc и т.д..

- Try features marked as EXPERIMENTAL (Попробовать функции, отмеченные как ЭКСПЕРИМЕНТАЛЬНЫЕ).

Данный пункт позволяет включить\исключить использование функций отмеченные как экспериментальный, данные функции не всегда могут работать, могут содержать ошибки или быть не завершённые.

- Debug crosstool-NG (Отладка crosstool-ng).

Включение отладочного режима для crosstool-NG. Данная опция обычно используется при разработке, илбо при исправлении возникших ошибок.

- Pause between every steps (Пауза между каждым шагом).

Сборка инструментария состоит из нескольких шагов и многие из них производятся в автоматическом режиме. Данная опция позволяет включить паузу между шагами для отладки и контроля процесса сборки.

- Save intermediate steps (Сохранять промежуточные шаги).

Данная опция позволяет сохранять шаги сборки для возможности продолжения повторного запуска сборки с конкретного шага. Данная опция полезна в случае необходимости прерывания процесса сборки либо при возникновении ошибок.

- Do `*not*` override `LC_MESSAGES` (EXPERIMENTAL) (Непереопределять `LC_MESSAGES`).

Данный пункт позволяет включить\отключить возможность записи сообщений на русском языке. По умолчанию crosstool-ng устанавливает и экспортирует `LC_ALL = C`, что сохраняет сообщения на английском языке. Так как данная опция экспериментальная, то её использование нежелательно.

- Interactive shell on failed commands (санг. «Интерактивная оболочка при неудачных командах»).

Данный пункт позволяет включить\отключить возможность создания интерактивной оболочки для каждой неудавшейся команды. Эта оболочка будет иметь ту же среду, с которой была выполнена неудачная команда, и рабочий каталог будет установлен в каталог, в котором была выполнена неудачная команда. После устранения проблемы можно выйти из интерактивной оболочки с любым из этих кодов выхода:

1. проблема была исправлена, продолжить сборку с помощью следующей команды;
2. проблема была исправлена, перезапустить неудачную команду;
3. прервать сборку.

- Local tarballs directory (Каталог локальных архивов).

В данном пункте прописывается путь к ранее загруженным архивам. Дело в том, что в процессе сборки инструментария производится скачивание необходимых архивов с исходными кодами. Они имеют довольно большой объем и требуется прямой доступ в сеть интернет. Когда отсутствует доступ в интернет, либо он лимитирован то можно производить сборку с использованием заранее скаченных пакетов.

- Save new tarballs (Сохранить новые архивы).

Данный пункт позволяет включить\отключить возможность сохранения новых загруженных архивов. Путь для их охранения указывается в параметре Local tarballs directory.

- Working directory (Рабочая директория).

В данном пункте прописывается путь к каталогу, в котором будут выполняться все действия по сборке. По умолчанию используется значение

`#{CT_TOP_DIR} /.Build"`, и если этот параметр пуст, он также будет использовать значение по умолчанию. Очень важным моментом является то что каталог сборки должен отличаться от каталога в котором хранятся исходные коды и от каталога в который после сборке будет произведена установка.

- Prefix directory (Префикс директории).

В данном пункте прописывается путь к каталогу, из которого будет работать кросс-компилятор. По сути это каталог установки, если данное значение не указано, то каталогом установки будет считаться корневой каталог гостевой системы.

- Remove documentation (с англ. «Удалить документацию»)

В данном пункте мы включаем\отключаем удаление установленной документации (основные и информационные страницы). В результате удаления освободится около 8MiB для компилятора на основе `uClibc`, `C` и `C++`.

- Render the toolchain read-only (Исполнять инструментарий только для чтения).

При включении данного пункта каталог установки инструментария и все его подкаталоги будут доступны только для чтения.

- Download agent (с англ. «Выбор утилиты загрузки»).

Данный пункт предлагает выбрать одну из утилит загрузки файлов (`wget` или `curl`).

- Forbid downloads (с англ. «Запрещать загрузки»).

В данном пункте устанавливается запрет на загрузку, так как правило, `crosstool-ng` будет пытаться загрузить отсутствующие архивы. Если сетевого подключение отсутствует, когда вы запускаете `crosstool-ng`, и некоторые файлы отсутствуют, то до появления ошибки может пройти много времени, прежде чем `crosstool-ng` завершится с ошибкой.

- Connection timeout (Время ожидания соединения).

В данном пункте устанавливается максимальное время в секундах, которое будет ожидаться ответ сервера.

- Extra options to wget\curl (Дополнительные опции для wget\curl).

В данном пункте устанавливаются дополнительные опции для работы утилиты загрузки архивов (`wget\curl`), например, указывается ограничение скорости загрузки или прокси сервер.

- Stop after downloading tarballs (с англ. «Останавливаться после загрузки архивов»).

Данная опция позволяет остановить процесс сборки сразу после загрузки архивов. Данная опция бывает необходима, когда сборка будет производиться на ПК без доступа в сеть интернет и требуется с другого компьютера скачать и перенести все необходимо для сборки.

- Use a mirror (Использовать зеркало).

Данная опция позволяет использовать зеркала для загрузки архивов с необходимыми исходными кодами.

- Only use mirror (Использовать только зеркало).

Данная опция указывает сборщику на то, что для скачивать необходимые архивы с исходными кодами стоит производить только с зеркальных источников.

- Number of parallel jobs (Количество параллельных процессов).

В данном пункте указывается количество параллельных процессов при сборке. Для максимально эффективного использования процессора при сборке количество параллельных процессов рассчитывается по формуле  $n+1$ , где  $n$  — это количество ядер(процессоров).

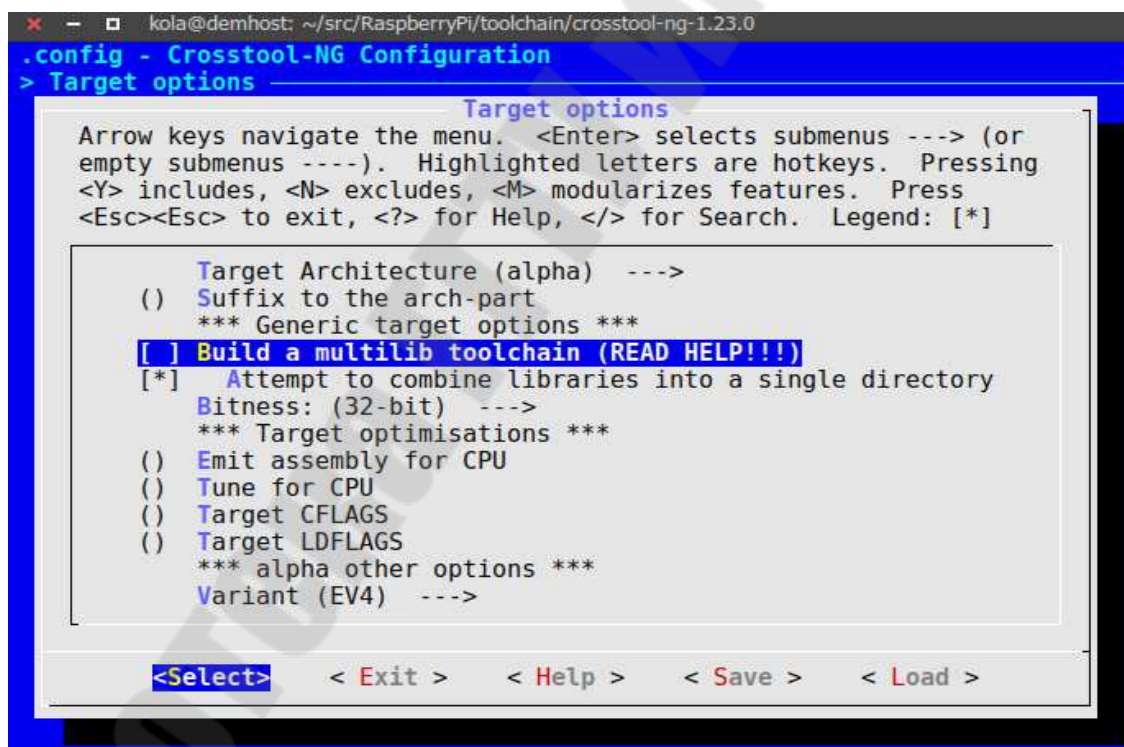
- Extra build compiler flags (Дополнительные флаги компиляции).

В данном пункте можно указать дополнительные флаги для передачи компилятору сборки C и C++.

- Extra build linker flags (Дополнительные флаги компоновщика сборки).

В данном пункте мы прописываем дополнительные флаги для передачи компоновщику сборки C и C++.

## Вкладка Target options (опции целевой архитектуры)



2.4

Рисунок 2.4 — Вкладка Target options

- Target Architecture (санг. «Целевая архитектура»).

В данном пункте мы выбираем архитектуру, для которой и будем собирать кросс-компилятор. На выбор предоставляются следующие варианты: alpha, arm, avr, m68k, mips, nios2, powerpc, sh, s390, sparc, x86, xtensa.

- Suffix to the arch-part (санг. «Суффикс к архитектуре»).

В данном пункте мы указываем специфичный суффикс для определенной архитектуры, так как много таких систем имеют разные варианты. Данный параметр не является обязательным.

- `Build a multilib toolchain` (санг. «Создание многофункционального инструментария»).

В данном пункте мы включаем\отключаем поддержку библиотеку C, оптимизированную для некоторых вариантов выбранной архитектуры, помимо настроек по умолчанию. Список вариантов зависит от архитектуры и является жестким кодом в gcc.

- `Attempt to combine libraries into a single directory` (Попытка объединить библиотеки в один каталог).

В данном пункте мы включаем\отключаем объединение библиотек в один каталог, так как не все пользователи данного инструментария могут обрабатывать библиотеки, находящиеся в нескольких каталогах. Чтобы удовлетворить их потребности, `crosstool-ng` может попытаться объединить библиотеки в один каталог `/lib` и создать все остальные каталоги в виде символических ссылок на `/lib`. Это требует, чтобы все имена библиотек были уникальными в каждом `sysroot`.

- `Bitness` (с англ. «Разрядность»).

В данном пункте мы выбираем разрядность архитектуры, для которой производится сборка.

- `Emit assembly for CPU` (санг. «Запустить сборку для CPU»).

В данном пункте мы указываем имя целевого процессора. GCC использует это имя, чтобы определить, какие команды он может запускать при генерации кода сборки. Это флаг конфигурации `--with-cpu = XXXX`, а флаг выполнения `-mcpu = XXX`. Выбирать значение нужно из руководства gcc для выбранной версии gcc и целевого процессора.

- `Tune for CPU` (санг. «Настройка для CPU»).

В данном пункте мы указываем параметр, который очень похож на предыдущий, за исключением того, что вместо указания фактического типа целевого процессора и, следовательно, ограничивая, какие инструкции можно использовать, он указывает, что GCC должен настраивать производительность кода так, как если бы цель имела тип указанный в этом параметре, но все же выбирая команды, которые он будет генерировать на основе процессора, указанного параметром выше или параметр (командной строки) `-mcpu=`. Это флаг конфигурации `--with-tune = XXXX`, а флаг выполнения `-mtune = XXX`. Выбирать значение нужно из руководства gcc для выбранной вами версии gcc и целевого процессора.

- `Target CFLAGS` (с англ. «Цель CFLAGS»).

В данном пункте мы указываем определенные опции при компиляции библиотек инструментария, которые будут выполняться на целевой архитектуре (например, `libc.so`). Нужно обратить внимание, что параметры,

указанные выше, будут автоматически использоваться. Нет нужды их указывать здесь.

- Target LDFLAGS (санг. «Цель LDFLAGS»).

В данном пункте мы указываем определенные опции при связывании библиотек собираемого инструментария, которые будут запускаться на целевой платформе.

## 2.5 Вкладка Toolchain options(Настройки инструментария)

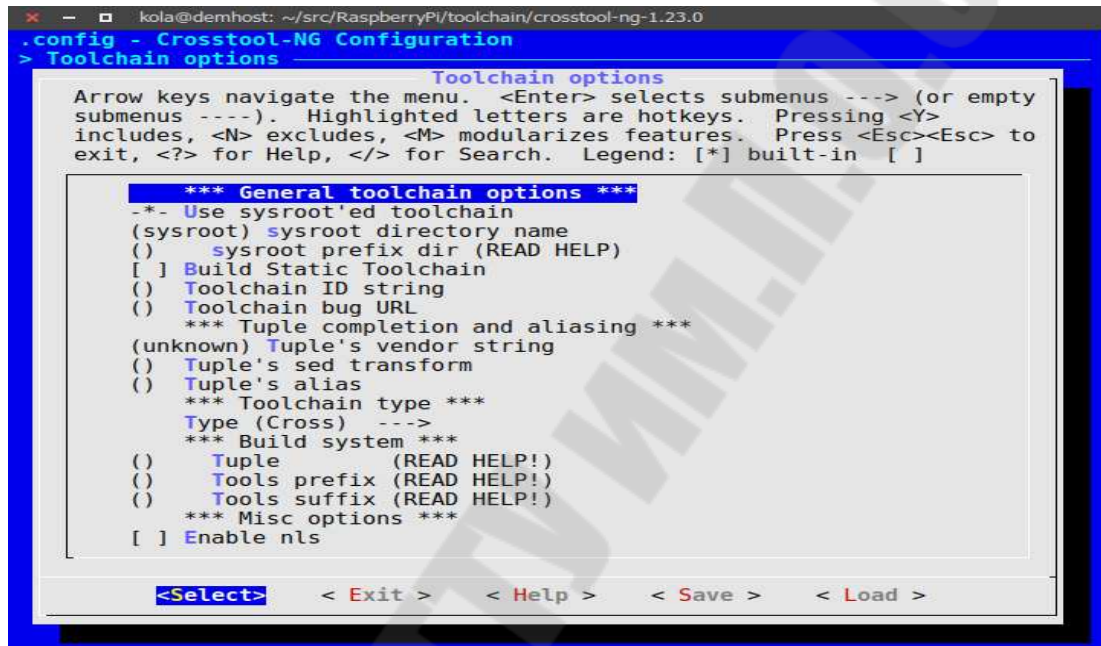


Рисунок 2.5 — Вкладка Toolchainoptions

- Usesysroot'edtoolchain(санг. «Использовать инструментарий sysroot»). Вданномпунктеговорится, чтомыбудемиспользоватьфункциюsysrootvgcc: библиотеки, разделенныемеждуprefix/target/sysroot/libипrefix/target/sysroot/usr/lib.

- Sysroot directory name (санг. «Название sysroot-каталога»).

В данном пункте задается имя каталога sysroot. Обычно это просто «sysroot» (по умолчанию) или «sys-root».

- Sysrootprefixdir (санг. «Префикс каталога sysroot»).

В данном задается префикс, который будет добавлен к пути sysroot, непосредственно перед фактическим каталогом sysroot. Фактически, путь sysroot построен как:

```
${CT_PREFIX_DIR}/${CT_TARGET}/${CT_SYSROOT_DIR_PREFIX}/${CT_SYSROOT_NAME}
```

- BuildStaticToolchain (санг. «Создание статического инструментария»).

В данном пункте активируется создание статического инструментария, для того, что бы была возможность переместить инструментарий на другую платформу, на котором могут отсутствовать необходимые версии системных библиотек, в таком случае все инструменты будут связаны статически.



- `ToolchainIDstring` (с англ. «Строка идентификатора инструментария»).

В данном пункте мы указываем строку, которая идентифицирует ваш пакет. Можно указать номер сборки или дату сборки. Эта строка версии будет включена в выход `gcc -version`, а также в `binutils`, `glibc`, `gdb` и `gdbserver`.

Если эта строка оставлена пустой, фактическая версия пакета будет:

```
"crosstool-NG $ {CT_VERSION}"
```

В противном случае это будет:

```
"crosstool-NG $ {CT_VERSION} - $ {CT_TOOLCHAIN_PKGVERSION}"
```

- `ToolchainbugURL` (с англ. «URL-адрес исправлений инструментария»).

В данном пункте задается URL-адрес, который могут посещать пользователи, если они хотят сообщить об ошибке.

- `Tuple's vendor string` (с англ. «`Tuple's vendor string`»).

В данном пункте задается производитель сборки системы. Для разделения слов используется одно слово или подчеркивание `"_"`. Запрещено использовать ни тире, ни пробел, так как это сможет создать проблемы.

- `Tuple'ssedtransform` (с англ. «Корректное преобразование `Tuple`»).

В данном пункте задается выражение, которое будет применено к `#{CT_TARGET}`, чтобы создать псевдоним для инструментария.

Например, `«s / $ {CT_TARGET_VENDOR}/foobar/»` (без двойных кавычек) создаст псевдоним (например) `armeb-foobar-linux-uclibc`. Не нужно ничего вводить здесь, если не планируется вручную вызывать инструменты (на основе `autotools ./configure` будет использоваться стандартное имя).

- `Tuple's alias` (с англ. «Псевдоним `Tuple`»).

В данном пункте задается строка, которая будет использоваться для создания символических ссылок на инструментальные средства `toolchain` (например, если здесь ввести `«foo-bar»`, то `gcc` для инструментария также будет доступен как `«foo-bar-gcc»` вместе с исходным именем). Не нужно ничего вводить здесь, если не планируется вручную вызывать инструменты (на основе `autotools ./configure` будет использоваться стандартное имя).

- `Type` (с англ. «Тип»).

В данном пункте мы выбирается тип создаваемого инструментария. Имеется два варианта выбора:

1. `Cross` - это классическая сборка, при которой ожидается, что созданный инструментарий будет запущен на том же компьютере, на котором и произведена сборка, и создаст код для запуска на другой платформе;

2. `Canadian` — вариант, при котором сборка будет производиться на одной платформе, запуск на другой, для работы на третьей.

- `Tuple`

В данном пункте задается каноническое название машины, создающей инструментарий.

- `Tools prefix` (с англ. «Префикс инструментов»).

В данном пункте задается префикс инструментов, которые находятся в необычном месте файловой системы.

- Tools suffix (с англ. «Суффикс инструментов»).

В данном пункте задается суффикс, например, gcc-4.3.2, если у уже есть gcc-4.3.1. В данном случае нужно ввести «-4.3.2».

- Enable nls (с англ. «Включить nls»).

В данном пункте активируется поддержка родного языка (native language support - nls).

- Tuple (с англ. «Tuple»).

В данном пункте задается каноническое название платформы, на которой установлен инструментарий.

- Tools prefix (с англ. «Префикс инструментов»).

В данном пункте задается префикс инструментов, которые находятся в необычном месте файловой системы на платформе, на которой установлен инструментарий.

- Tools suffix (с англ. «Суффикс инструментов»).

В данном пункте задается префикс инструментов платформы, на которой установлен инструментарий, например, gcc-4.3.2, если у уже есть gcc-4.3.1. В данном случае нужно ввести «-4.3.2».

## 2.6 Вкладка Operating System (Настройка операционной системы)

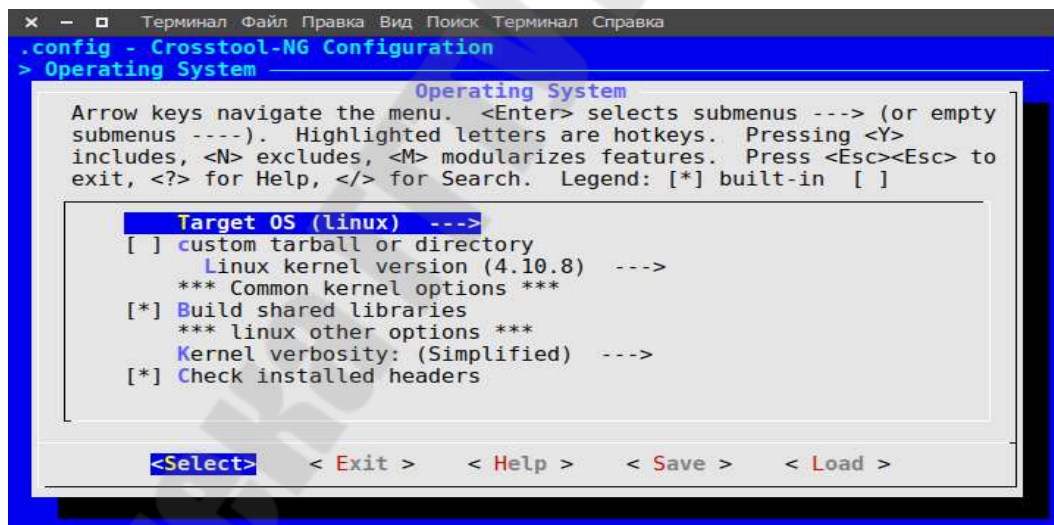


Рисунок 2.5 - Подпункты настроек операционной системы

- Target OS (с англ. «Целевая операционная система»).

В данном пункте выбирается операционная система целевой платформы:

1. bare-metal — данный вариант относится для тех программ, которые работают без операционной системы, на «голом железе»;
2. linux — вариант, при котором сборка инструментария будет построена для системы работающая на ядреLinux.

- Customtarballordirectory (с англ. «Пользовательский архив или каталог»).

Данную опцию требуется включить если выбранная версия Linux не загружается. Вместо этого мы можно использовать произвольное местоположение, чтобы получить исходники.

- Path to custom source, tarball or directory (англ. «Путь к пользовательским исходникам, архиву или директории»).

В данном пункте мы задается путь к каталогу или архиву исходников для linux. Если путь является архив, то он должен содержать: <имя> - <версия> /, где имя этого компонент - linux, а версия приведена ниже в строке пользовательской версии.

- CustomLinuxversion(с англ. «Пользовательская версия Linux»).

В данном пункте задается номер версии linux.

- Buildsharedlibraries(с англ. «Создание общих библиотек»).

В данном пункте отключается создание динамических библиотек.

- Kernel verbosity (с англ. «Переменные ядра»).

В данном пункте мы выбираем стиль команд:

1. Simplified — печать упрощенных команд;
2. Full commands — печать полных команд;
3. Exec reasons — печать причины, по которым восстанавливается цель make.

- Checkinstalled headers (с англ. «Проверка установленные заголовки»).

В данном пункте активируется проверка установленных заголовков, если есть сомнения, что установка была ошибочна.

## 2.7 Вкладка Binary utilities (Настройка бинарных утилит)

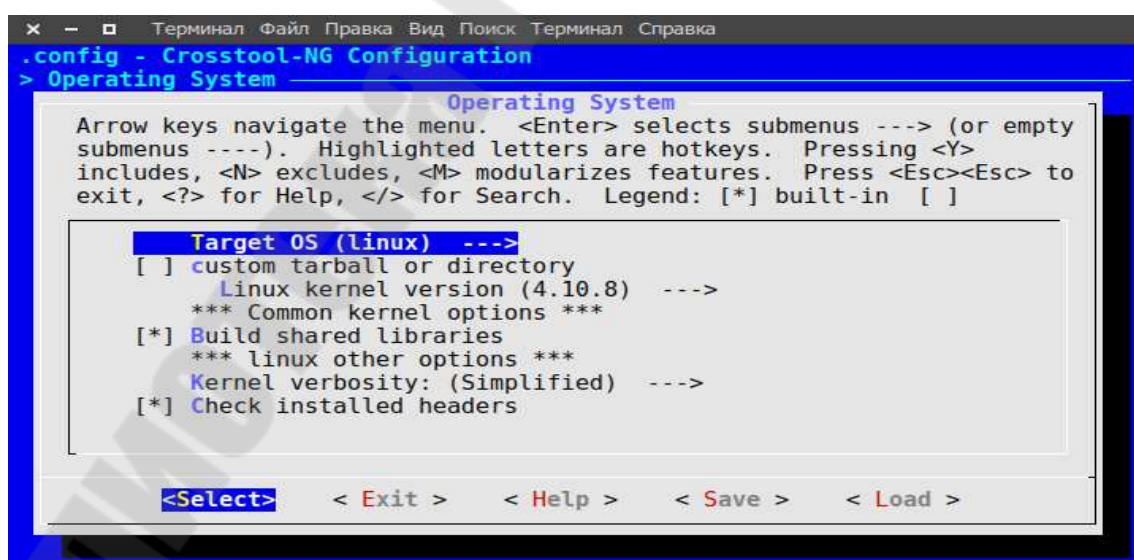


Рисунок 2.6 - подпункты настроек бинарных утилит

- Binary format (с англ. «Бинарный формат»).

Данная настройка позволит целевой системе создавать исполняемые файлы ELF, подходящие для архитектур с MMU.

- Binutils

Подключение бинарной утилиты Binutils.

- Show Linaro versions (с англ. «Показать версии Linaro»).

В данном пункте активируется Linaro, который поддерживает некоторые продвинутые (более стабильные) экспериментальные версии binutils, особенно для архитектуры ARM.

- Binutils version (с англ. «Версия binutils»).

В данном пункте выбирается версия binutils

- Linkerstoenable(с англ. «Включение компоновщика»).

В данном пункте выбирается компоновщик:

1. ld — исторический, bfd-компоновщик;
2. ld, gold — будет установлен как исторический ld, так и новый gold компоновщик, а ld - используемый компоновщик по умолчанию;
3. gold, ld — будет установлен как исторический ld, так и новый gold компоновщик, а gold - используемый компоновщик по умолчанию.

- Enable threaded gold (с англ. «Включить потоковое gold»).

В данном пункте активируется работа с потоками.

- Addldwrapper (с англ. «Добавить ld-обертку»).

В данном пункте активируется добавление ld-обертки, которая вызывает либо gold, либо ld. По умолчанию оболочка будет вызывать обертку по умолчанию, но если установить переменную окружения CTNG\_LD\_IS, то можно изменить, какой компоновщик будет вызван:

CTNG\_LD\_IS = gold - безоговорочно называют gold компоновщик

CTNG\_LD\_IS = bfd - безоговорочно вызовет старый bfd ld-компоновщик.

- Enablesupportforplugins(с англ. «Включить поддержку плагинов»).

В данном пункте мы активируется поддержка плагинов. В частности, gold может использовать lto-plugin, как установлено gcc, для обработки LTO.

- Binutilsextraconfig(с англ. «Дополнительная настройка binutils»).

Здесь задаются дополнительные флаги и настройки. Можно ввести несколько аргументов, и аргументы могут содержать пробелы, если они правильно указаны (или экранированы, предпочтительно кавычки). Например: --with-foo = "1й аргумент с 4 пробелами" --with-bar = 2й-аргумент-без-пробелов.

- Binutils libraries for the target (с англ. «Библиотеки binutils для целевой системы»).

В данном пункте активируется поддержку библиотек binutils, в которых могут нуждаться некоторые утилиты, например, orgprofile.

- Libbfd.

В данном пункте мы подключаем библиотеку libbfd.

- Libiberty.

В данном пункте мы подключаем библиотеку libiberty.

## 2.8 Вкладка C-library (Конфигурация библиотек языка C)

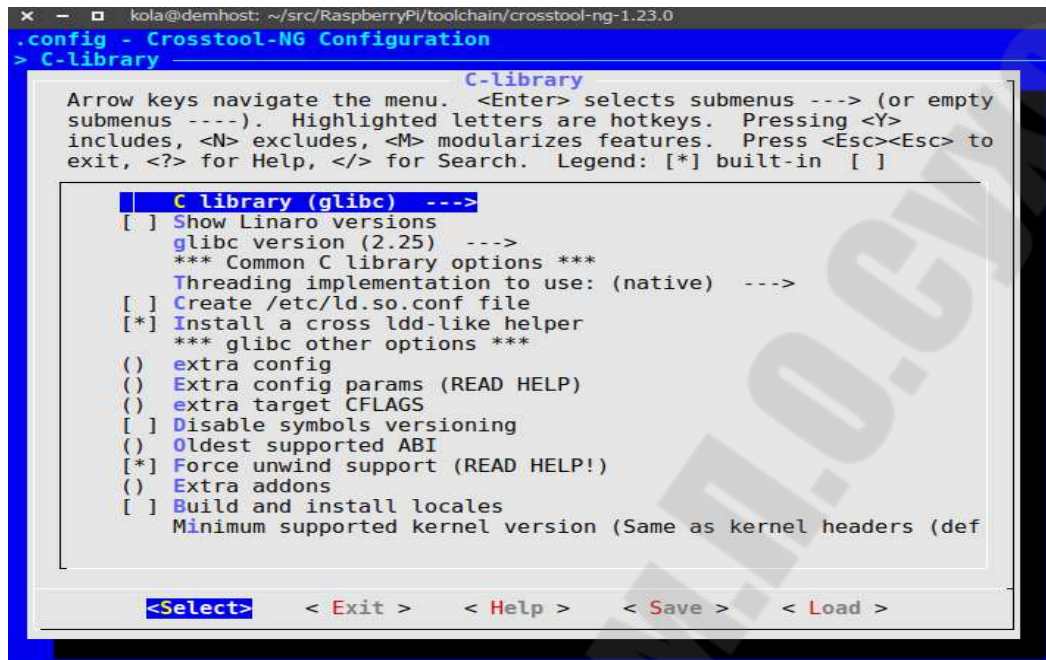


Рисунок 2.7. Подпункты конфигурации библиотек языка C

- C library (с англ. «Библиотека Си»).

В данном пункте выбирается какая именно библиотека будет использоваться:

1. glibc — де-факто стандарт для дистрибутивов Linux. Многофункциональная, но большая... Самая полезная для настольных систем;
2. uClibc — де-факто стандарт для встроенных систем Linux. Большая конфигурация, без ущерба для размера.

- Show Linaro versions (с англ. «Показывать версию Linaro»).

В данном пункте активируется отображение версии Linaro.

- glibc version (с англ. «Версия glibc»).

В данном пункте выбирается версия glibc, которая будет использоваться в кросскомпиляторе.

- Threading implementation to use (с англ. «Реализация поточности»).

В данном пункте выбирается реализация собственных потоков для выбранной системы и библиотеки C. Например, в Linux с glibc это NPTL; на Windows с mingw это win32.

- Create /etc/ld.so.conf file (с англ. «Создать файл /etc/ld.so.conf»).

В данном пункте активируется создание файла /etc/ld.so.conf в каталоге sysroot для целевой системы.

- Extra directories to add to /etc/ld.so.conf (с англ. «Дополнительные каталоги для добавления в /etc/ld.so.conf»).

Здесь вводятся дополнительные каталоги для включения в /etc/ld.so.conf. Каталоги будут дополнены спецификацией multilibs компилятора, если это применимо. Например, если multilibs компилятора включает /lib и /lib64, и

будет указано здесь /usr/local/lib, ld.so.conf будет иметь как /usr/local/lib, так и /usr/local/lib64.

- Install a cross ldd - like helper (с англ. «Установить cross-ldd в качестве помощника»).

В данном пункте включается установка помощника, подобного ldd, который можно запустить в своей целевой системе, и он будет (пытаться) разрешать зависимости разделяемых библиотек, как если бы они выполнялись на целевой системе.

- extra config (с англ. «Дополнительные настройки»).

Здесь задаются дополнительные флаги для ./configure при настройке. Здесь можно вводить несколько аргументов, и аргументы могут содержать пробелы, если они правильно указаны (с использованием кавычек, или дефисов). Например: --with-foo = "1й аргумент с 4 пробелами" --with-bar = 2й-аргумент-без-пробелов.

- Extraconfigparams(с англ. «Дополнительные параметры конфигурации»).

Здесь задаются некоторые параметры для целевых архитектур в файле configparms. Это относится к sh3/4, для которой действительно необходимо установить configparms в «no-z-defs = yes» по состоянию на gcc-3.4/glibc-2.3.2. Если создается инструментарий не для sh3/4, то данную опцию лучше не включать. Если нужно передать более одного значения, то требуется отделить их «\n». Например: var1 = val1 \n var2 = val2.

- extratargetCFLAGS (с англ. «Дополнительные настройка CFLAGS»).
- Disable symbols versioning (с англ. «Отключить символы версий»).

В данном пункте отключается включение информации о версиях в объектах библиотеки.

- OldestsupportedABI(с англ. «Самый старый поддерживаемый ABI»).

Здесь задается самый старый ABI, поддерживаемый библиотекой C. Если этот параметр не установлен, (e)glibc выберет самостоятельно.

- Forceunwindsupport(с англ. «Принудительная поддержка»).

В данном пункте включается принудительная поддержка, если инструментарий не работает при создании файлов начальной загрузки библиотеки C или полной библиотеки C с сообщением типа: configure: error: принудительная перезагрузка необходима, то можно попробовать изменить эту опцию.

- Extra addons (с англ. «Дополнительные дополнения»).

Здесь задаются дополнения, которые должны быть разделены пробелами.

- Buildandinstalllocales(с англ. «Создание и установка локалей»).

В данном пункте активируется создание и установка файлов локали libc для целевой системы, для поддержки интер-национализации.

- Minimum supported kernel version (с англ. «Минимальная поддерживаемая версия ядра»).

В данном пункте мы задается минимальная поддерживаемая версия ядра:

1. Let `./configure decide` — пусть `./configure` решит, с какой минимальной версией ядра `glibc` будет работать. Это будет включать устаревший код совместимости для старых ядер в библиотеке `C`, гарантируя, что он будет работать на большом количестве старых ядер;

2. `Same as kernel headers` — вариант, чтобы `glibc` запускался с той же версией ядра, что и заголовки. Это значение по умолчанию. Если включено, `crosstool-ng` будет использовать выбранную версию заголовков ядра для поддерживаемой минимальной версии ядра `glibc`, которая передается в «`--enable-kernel =>`» при настройке `glibc`. Включение этого гарантирует, что в библиотеки `C` не будет добавлен старый код совместимости для старых ядер, но он не сможет работать на версиях ядра, более ранних, чем любая версия заголовков ядра, для которой создан инструментарий;

3. `Specific kernel version` — указать самую раннюю версию ядра `Linux`, на которой требуется включить поддержку `glibc`. Она не должна соответствовать версии заголовков ядра, используемой для инструментария. Он контролирует, что передается опции «`--enable-kernel =>`» в `glibc configure` скрипт. Если вы хотите, чтобы у вас была возможность статически связывать программы с библиотекой `C` вашего инструментария, убедитесь, что эта версия ядра ниже всех ядер, которые вы хотите поддерживать, чтобы избежать ошибок «`FATAL: kernel too old`». Чем выше указанная вами версия, тем менее унаследованный код будет встроен в `libc`.

- `Minimum kernel version to support` (санг. «Минимальная версия ядра для поддержки»).

Здесь задается самую низкую версию ядра `glibc`, с которой можно будет работать.

- `uClibc version` (с англ. «Версия `uClibc`»).

В данном пункте выбирается версия библиотеки языка Си.

- `uClibc verbosity` (с англ. «`uClibc` команды»).

В данном пункте задается вариант печати команд `uClibc`:

1. *Quiet build* — печать коротких команд;
2. *Brief build* — печать упрощенных команд;
3. *Very verbose build* — печать полных команд.

- `Configuration file` (с англ. «Конфигурационный файл»).

Здесь задается путь к файлу конфигурации. Если файл не указан, то используется файл по умолчанию.

- `Addsupportforlocales` (с англ. «Добавить поддержку для локалей»).

В данном пункте активируется поддержка в `uClibc` локализации.

- `AddsupportforIPv6` (с англ. «Добавить поддержку для IPv6»).

В данном пункте активируется поддержка в `uClibc` IPv6.

- `AddsupportforWCHAR` (с англ. «Добавить поддержку WCHAR»).

В данном пункте включается поддержка в `uClibc` WCHAR.

- `Addsupportforfenv.h` (с англ. «Добавить поддержку `fenv.h`»).

В данном пункте включается поддержка в uClibc fenv.h. Он предоставляет функции для управления средой с плавающей точкой, такие как режим округления, исключения и т. д. Для некоторых архитектур fenv.h является неполным, поэтому он не установлен по умолчанию. Известно, что x86 имеет довольно полный файл fenv.h, поэтому он устанавливается по умолчанию только для x86.

- AddsupportforRPC(санг. «Добавить поддержку RPC»).

В данном включается поддержка в uClibc удаленных вызовов процедур (RPC).

- Use -uclicsgnueabisuffix(санг. «Использовать суффикс -uclicsgnueabi»).

В данном пункте активируется использование суффикса -uclicsgnueabi. В зависимости от того, где будет использоваться инструментарий, может потребоваться настроить «системную» часть. Buildroot предпочитает иметь arm-\*-linux-uclicsgnueabi; OpenEmbedded предпочитает arm-\*-linux-uclicseabi. Другие инструменты, либо принимают оба, либо не заботятся о суффиксе.

## 2.9 Вкладка C compiler (Настройка компиляции языка C)

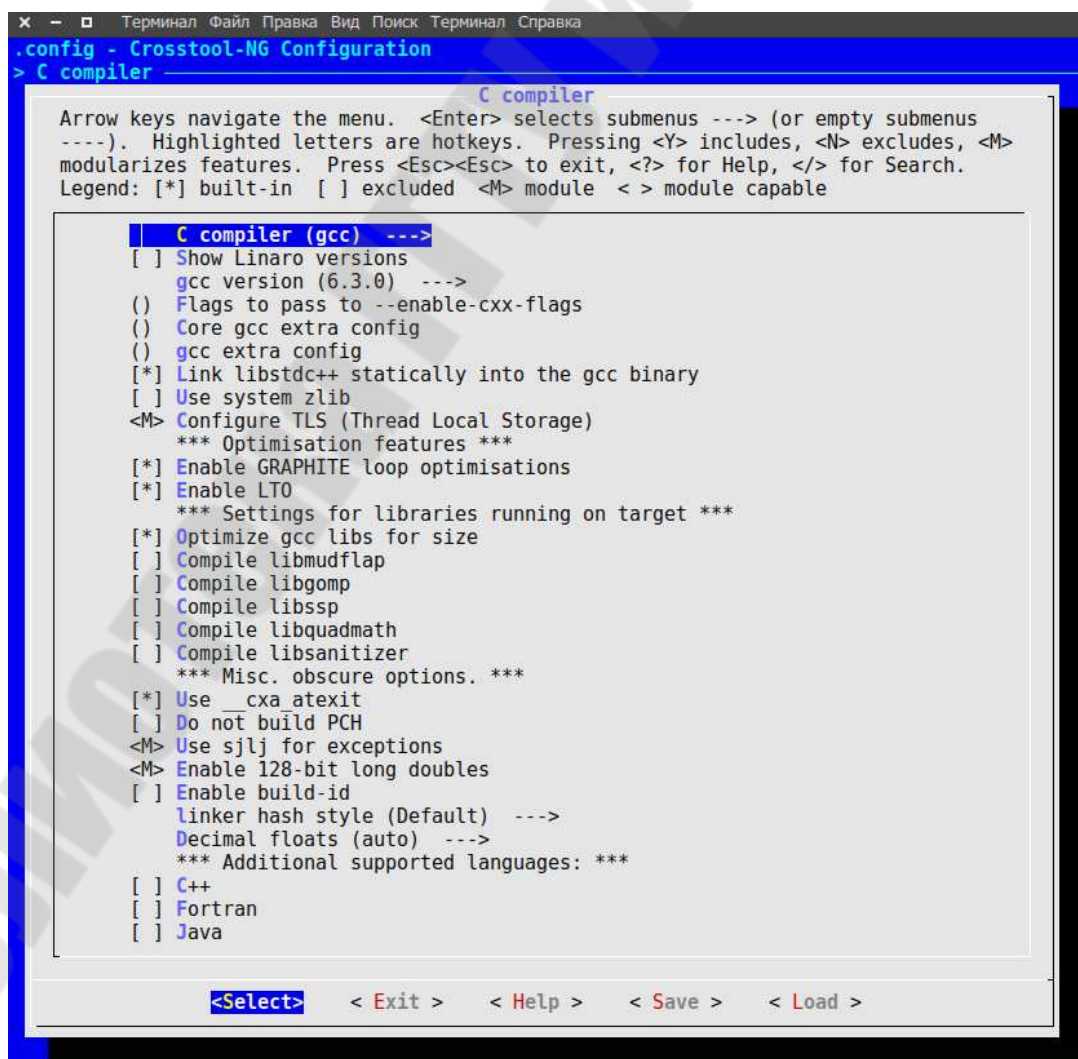


Рисунок 2.8 - подпункты настройки компиляции языка C



- C compiler (с англ. «Си компилятор»).

В данном пункте задается си компилятор.

- Show Linaro versions (с англ. «Показывать версию Linaro»).

В данном пункте включается отображение версии Linaro.

- gcc version (с англ. «Версия gcc»).

В данном пункте задается версия gcc.

- Flags to pass to --enable-cxx-flags (с англ. «Флаги для перехода на --enable-cxx-flags»).

Здесь задаются значение параметра gcc ./configure --enable-cxx-flags.

- Coregcssextraconfig(сангл. «Конфигурация ядра gcc»).

Здесь задаются дополнительные флаги для перехода на ./configure при настройке ядра gcc. Ядро gcc - это сжатый, C-единственный компилятор, необходимый для сборки библиотеки C.

- gcssextraconfig(сангл. «Дополнительная конфигурация gcc»).

Здесь задаются дополнительные флаги для перехода на ./configure при настройке gcc.

- Link libstdc++ statically into the gcc binary (сангл. «Статическая ссылка libstdc++ в двоичный файл gcc»).

В данном пункте активируется создание статической ссылки libstdc ++, так как для новых версий gcc требуются некоторые библиотеки с ++. Такая ссылка увеличивает вероятность того, что двоичный файл gcc будет работать на машинах, отличных от тех, на которых он был построен, и не беспокоиться о распространении соответствующей версии libstdc ++ вместе с ней.

- Usesystemzlib(с англ. «Использовать системный zlib»).

В данном пункте активируется использование системного zlib.

- Enable GRAPHITE loop optimisations (сангл. «Включить оптимизацию цикла GRAPHITE»).

В данном пункте активируется оптимизация цикла GRAPHITE.

- Enable LTO (с англ. «Включить LTO»).

В данном пункте активируется оптимизация времени связи.

- Optimize gcc libs for size (сангл. «Оптимизировать gcc библиотеке по размеру»).

В данном пункте активируется оптимизация размеров библиотек.

- Compile libmudflap (с англ. «Компиляция libmudflap»).

В данном пункте активируется компиляция libmudflap – инструмент проверки использования указателя, который может обнаруживать различные неправильные применения указателей в C и (до некоторой степени) C ++.

- Compile libgomp (с англ. «Компиляция libgomp»).

В данном пункте включается компиляция libgomp, который является реализацией GNU интерфейса программирования приложений OpenMP (API) для многоплатформенного параллельного программирования с параллельной платформой C/C++ и Fortran.

- Compile libssp (с англ. «Компиляция libssp»).

В данном пункте включается компиляция `libssp` – это библиотека защиты от вредоносных программ.

- `Compile libquadmath` (с англ. «Компиляция `libquadmath`»).

В данном пункте включается компиляция `libquadmath` – это библиотека, которая предоставляет математические функции с четкой точностью на объектах, поддерживающих тип данных `__float128`.

- `Compile libsanitizer` (сангл. «Компиляция `libsanitizer`»).

В данном пункте активизируется компиляция `libsanitizer` – это библиотека, которая обеспечивает временную санитацию одного или обоих из:

- шаблоны доступа к памяти (вне связи, без использования)
- надежный доступ к данным (в многопоточных программах).

- `Enable build-id` (сангл. «Включить идентификатор сборки»).

В данном пункте задается GCC передача опции `--build-id` в компоновщик для всех конечных ссылок (ссылки, выполняемые без опции `-r` или `--relocatable`), если компоновщик поддерживает ее. Если компоновщик не поддерживает параметр `--build-id`, выдается предупреждение, и этот параметр игнорируется.

- `linker hash style` (сангл. «Стиль хэша компоновщика»).

В данном пункте выбирается стиль хэша компоновщика:

1. `Default` — не указывать какое-либо значение и использовать значение по умолчанию (`sysv`);
2. `sysv` — использовать синтаксический стиль `SYSV`;
3. `gnu` — использовать синтаксический стиль `GNU`;
4. `both` — использовать стиль как `GNU`, так и `SYSV`.

- `Decimal floats` (с англ. «Десятичная плавающая точка»).

В данном пункте задается тип десятичной плавающей точки:

1. `auto` — позволить `./configure` решить самому;
2. `bid` — использовать формат «двоичный целочисленный десятичный»;
3. `dpd` — использовать десятичную запятую для десятичных запятых;
4. `no` — не поддерживать десятичные запятые.

- `C++`

В данном пункте включается создание компилятора `C++`.

- `Fortran`

В данном пункте включается создание компилятора `Fortran`.

- `Java`

В данном пункте включается создание компилятора `Java`.

## 2.10 Вкладка Debug facilities (Подключение отладочных средств)

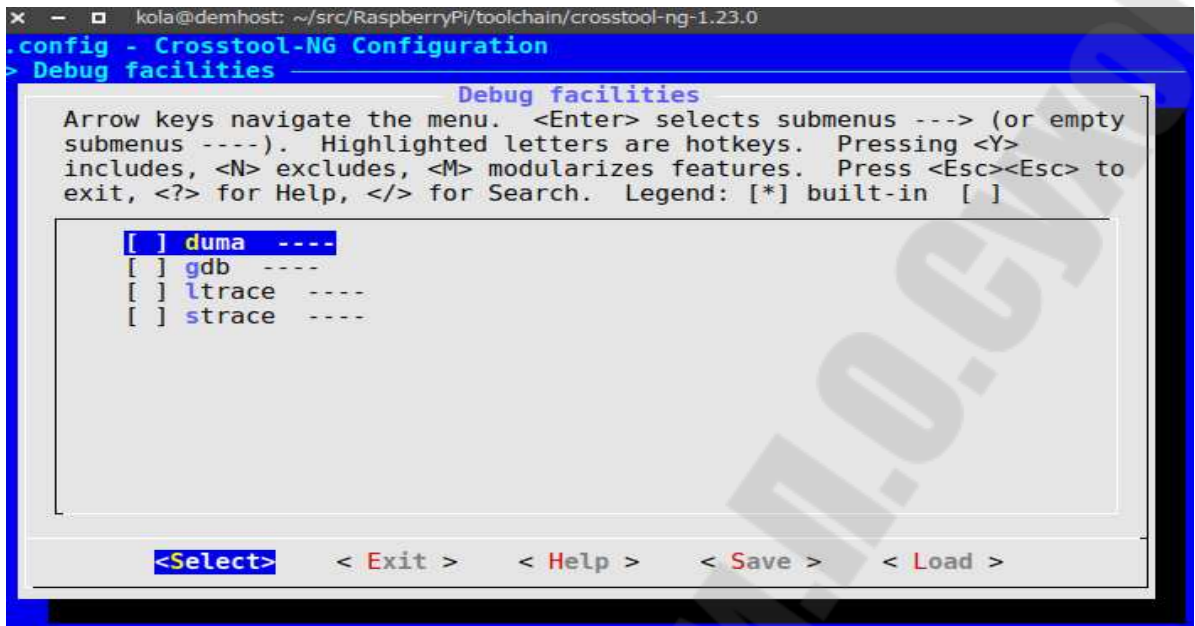


Рисунок 2.8. Подпункты настройки подключения отладочных средств

- duma

В данном пункте активируется использование duma. D.U.M.A. - Обнаружение непреднамеренного доступа к памяти. Проверка привязки памяти с дополнительными функциями. Ранее известный как электрический забор.

- gdb

В данном пункте включается сборка отладчика GNU.

- ltrace

В данном пункте включается поддержка ltrace. ltrace - это программа, которая просто запускает указанную команду до ее выхода. Он перехватывает и записывает вызовы динамической библиотеки, вызываемые выполненным процессом, и сигналы, полученные этим процессом.

Он также может перехватывать и распечатывать системные вызовы, выполняемые программой.

- strace (с англ. «strace»).

В данном пункте включается использование легкого отладчика strace.

- Build a shared library (с англ. «Создание общей библиотеки»).

В данном пункте включается создание общей библиотеки.

- Install custom D.U.M.A wrapper (с англ. «Установить выборочную обертку D.U.M.A»).

В данном пункте включается установку выборочной обертки D.U.M.A.

- D.U.M.A. version (с англ. «Версия D.U.M.A.»).

В данном пункте задается версия D.U.M.A.

- Cross-gdb (с англ. «Кросс-отладчик gdb»).

В данном пункте включается создание и установка кросс-отладчика gdb для целевой платформы.

- `Buildstaticcrossgdb` (с англ. «Создание статического кросс-отладчика `gdb`»).

В данном пункте включается создание статического кросс-отладчика `gdb` для целевой платформы, если нужно отлаживать платформу, которая не является той, которая используется для компиляции инструментария.

- `Enable'sim'` (с англ. «Включить `'sim'`»).

В данном пункте включается эмулятор `sim`.

- `Enablepythonscripting` (с англ. «Включить скрипты `python`»).

В данном пункте включаются скрипты `python`.

- `Pythonbinarytouse` (с англ. «Использовать двоичный код `Python`»).

Здесь задается путь к двоичному файлу `Python`, если он недоступен под именем по умолчанию (т.е. «`Python`»). По умолчанию `crosstool-ng` будет пытаться использовать `python`, `python3` и `python2` в этом порядке.

- `Cross-gdb extra config` (с англ. «Дополнительная настройка кросс-отладчика `gdb`»).

Здесь задаются дополнительные флаги для передачи в `./configure` при настройке кросс-отладчика `gdb`.

- `Native gdb`.

В данном пункте активируется создание и установка собственного `gdb` для целевой платформы.

- `gdbserver`.

В данном пункте включается создание и установка `gdbserver` для целевой платформы.

- `Build the IPA library` (с англ. «Создание библиотеки `IPA`»).

В данном пункте включается использование точки трассировки при отладке своих программ с помощью `gdbserver`.

- `gdb version` (с англ. «Версия `gdb`»).

В данном пункте выбирается версия `gdb`-отладчика.

- `ltrace version` (с англ. «Версия `ltrace`»).

В данном пункте выбирается версия `ltrace`.

- `strace version` (с англ. «Версия `strace`»).

В данном пункте выбирается версия `strace`.

## 2.11 Вкладка Companion libraries (Подключение сопутствующих библиотек)

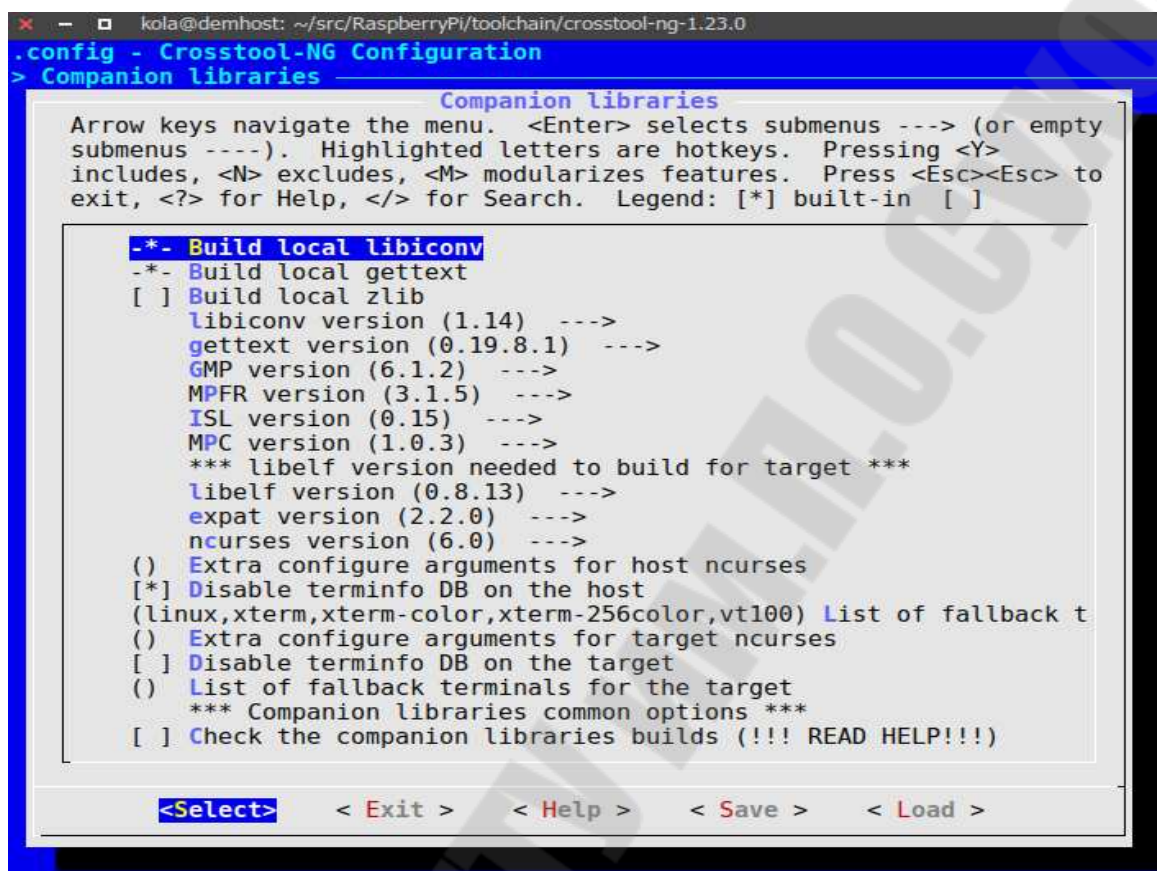


Рисунок 2.9. Подпункты настройки подключения сопутствующих библиотек

- Buildlocallibiconv (с англ. «Построить локальный libiconv»).  
В данном пункте включается сборка libiconv.
- Build local gettext (с англ. «Создание локального gettext»).  
В данном пункте включается сборка gettext.
- Buildlocalzlib (с англ. «Построить локальный zlib»).  
В данном пункте включается сборка zlib.
- zlib version (с англ. «Версия zlib»).  
В данном пункте задается версия zlib.
- libiconv version (с англ. «Версия libiconv»).  
В данном пункте выбирается версия libiconv.
- gettext version (с англ. «Версия gettext»).  
В данном пункте задается версия gettext.
- GMP version (с англ. «Версия GMP»).  
В данном пункте выбирается версия.
- MPFR version (с англ. «Версия MPFR»).  
В данном пункте выбирается версия MPFR.
- ISL version (с англ. «Версия ISL»).  
В данном пункте выбирается версия ISL.
- MPC version (с англ. «Версия MPC»).

В данном пункте выбирается версия MPC.

- libelf version (санг. «Версия libelf»).

В данном пункте выбирается версия libelf.

- ncurses version (санг. «Версия ncurses»).

В данном пункте выбирается версия ncurses.

- expat version санг. «Версия expat»).

В данном пункте выбирается версия expat.

- Extra configure arguments for host ncurses (с англ. «Дополнительные параметры конфигурации для платформы»).

Здесь задаются дополнительные аргументы, переданные дословно при настройке при создании хостов ncurses.

- Disable terminfo DB on the host (санг. «Отключить базу данных terminfo на хосте»).

В данном пункте разрешается встроить в библиотеку некоторые популярные определения терминалов. В настоящее время база данных terminfo не установлена на хосте в составе инструментария, созданного crosstool-ng. Это означает, что библиотека не сможет использовать терминалы, если эта база данных не предустановлена.

- List of fallback terminals for the host (с англ. «Список резервных терминалов для хоста»).

Здесь вводится список описаний терминалов, которые будут скомпилированы в библиотеку curses для хоста.

- Extra configure arguments for target ncurses (с англ. «Дополнительные параметры конфигурации для целевой системы ncurses»).

Здесь задаются дополнительные аргументы, переданные дословно при настройке при создании целевых ncurses.

- Disable terminfo DB on the target (с англ. «Отключить базу данных terminfo на целевой системе»).

В данном пункте отключается база данных terminfo на целевой системе.

- List of fallback terminals for the target (с англ. «Список резервных терминалов для целевой системы»).

Здесь вводится список описаний терминалов, которые будут скомпилированы в библиотеку curses для целевой системы.

- Check the companion libraries builds (с англ. «Проверить сборку сопутствующих библиотек»).

В данном пункте включается проверка сборки сопутствующих библиотек.

## 2.12 Вкладка Companion tools (Подключение сопутствующих инструментов)

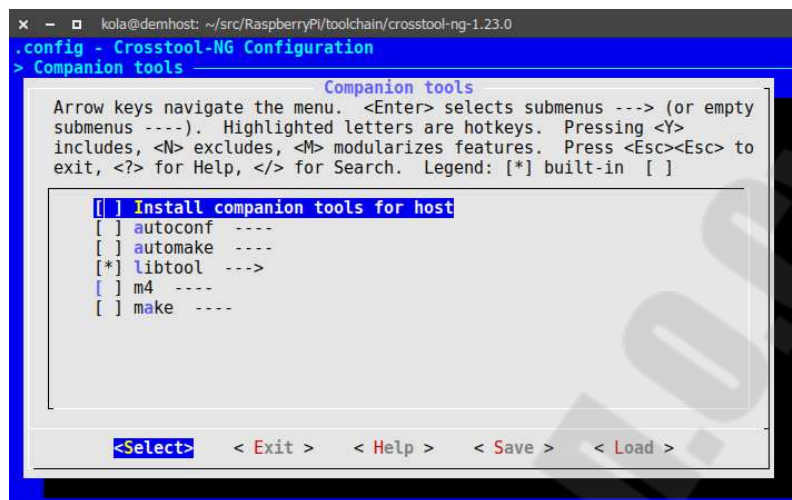


Рисунок 2.10. Подпункты подключения сопутствующих инструментов

- Install companion tools for host (санг. «Устанавливать вспомогательные инструменты для хоста»).

В данном пункте включаются вспомогательные компоненты и инструменты с добавлением их в инструментарий.

- autoconf

В данном пункте включается autoconf.

- automake

В данном пункте мы включается automake.

- libtool

В данном пункте включается libtool.

- m4.

В данном пункте мы включается GNU m4.

Для включения\исключения данного пункта нажать клавишу <Y>\<N>.

- make.

В данном пункте включается GNU make.

- Autoconf version.

В данном пункте задается версия Autoconf.

- Automake version (с англ. «Версия automake»).

В данном пункте задается версия automake.

- Libtool version (с англ. «Версия libtool»).

В данном пункте задается версия libtool.

- m4 version (с англ. «Версия m4»).

В данном пункте выбирается версия m4.

- Make version (с англ. «Версия make»).

В данном пункте выбирается версия make.

- Add gmake symlink to companion gnu/make (с англ. «Добавить символическую ссылку gmake в вспомогательный gnu/make»).

В данном пункте активируется добавление символической ссылки gmake в вспомогательный gnu/make.

### 3. Порядок выполнения работы

Сначала рассмотрим примеры сборки кросскомплета по заданную архитектуру. В качестве основной платформы (той на которой будут производиться все манипуляции) будет использоваться обычный персональный компьютер с X86 совместимой архитектурой процессора.

#### 3.1 Сборка crosstool-NG и кросскомпилятора для Raspberry Pi

Как уже было сказано, для получения исходного кода пакета необходимо произвести клонирование с сервера GitHub:

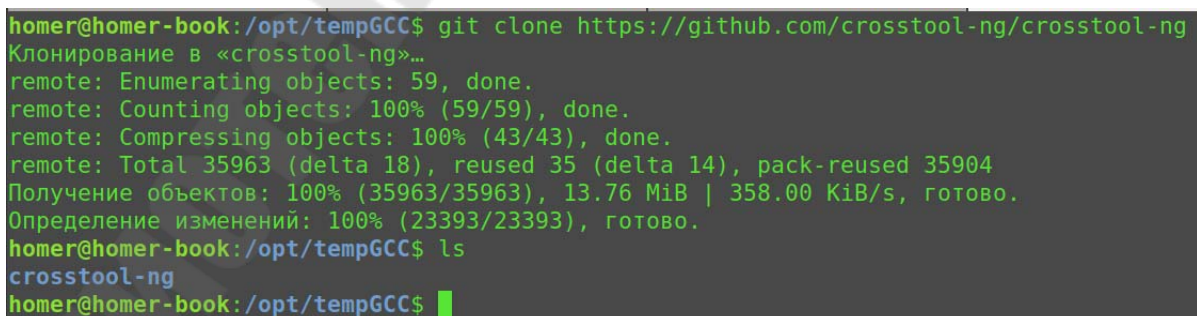
```
git clone https://github.com/crosstool-ng/crosstool-ng
```

При удачном клонировании в текущей папке папка с исходным кодом crosstool-ng. По умолчанию клиент git обычно не установлен в системе, поэтому при выполнении команды клонирования может возникнуть ошибка. И следует установить клиент git и повторить клонирование. Для установки используется следующая команда:

```
sudo apt install git git-gui
```

где git — это консольный клиент, а git-gui — это клиент, имеющий графический интерфейс пользователя. Для запуска клиента с графическим интерфейсом надо выполнить команду git gui.

Процесс клонирования исходного кода приведет на рисунке 2.11.



```
homer@homer-book:/opt/tempGCC$ git clone https://github.com/crosstool-ng/crosstool-ng
Клонирование в «crosstool-ng»...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 35963 (delta 18), reused 35 (delta 14), pack-reused 35904
Получение объектов: 100% (35963/35963), 13.76 MiB | 358.00 KiB/s, готово.
Определение изменений: 100% (23393/23393), готово.
homer@homer-book:/opt/tempGCC$ ls
crosstool-ng
homer@homer-book:/opt/tempGCC$
```

Рисунок 2.11 — Процесс клонирования исходного кода

После получения исходного кода необходимо установить все зависимости для сборки crosstool-ng. Как уже было сказано выше данный пакет распространяется в виде исходного кода, то для сборки потребуются дополнительные пакеты. Так же стоит заметить, что последние версии



дистрибутивы Ubuntu поставляются без установленных пакетов make и gcc, поэтому стоит добавить данные пакеты в список зависимостей.

Список необходимых пакетов:

- gcc, g++
- make
- mercurial
- bison
- cvs
- gperf
- flex
- texinfo
- automake
- build-essential
- libtool
- libncurses-dev

Все перечисленные пакеты можно установить с помощью одной команды:

```
sudo apt install gcc g++ make mercurial bison cvs gperf flex texinfo automake build-essential libtool libncurses-dev
```

Затем по выше описанному алгоритму надо приступить к сборке crosstool-ng. Стоит отметить что по правилам хорошего тона в Linux для сборки пакетов из исходных кодов стоит использовать папку, отличную от папки с исходным кодом. Это связано с различными причинами, например, в папке исходных кодов могут находиться файлы, одноименные с тем что должны будут созданы в процессе сборки, что в свою очередь приведет к ошибке. Или банальная ситуация с «засорением» исходного кода файлами сборки, которые в последствии будет крайне тяжело вычистить. Поэтому для сборки стоит создать отдельную папку, пусть она будет иметь имя build\_crosstool-ng и находиться в той же папке, где и папка с исходным кодом. Создаем необходимую папку:

```
mkdir build_crosstool-ng  
cd build_crosstool-ng
```

После того как папка создана и произведен переход в нее, можно приступить к конфигурации пакета. Для конфигурации используется следующая команда:

```
../crosstool-ng/configure --prefix=/opt/crosstool-ng
```

где параметр --prefix задает папку для установки собранного пакета crosstool-ng. Если данный параметр не будет задан, то установка будет

производиться в корневую файловую систему, что в свою очередь приведет к «засорению» операционной системы. Данная опция присутствует в скриптах конфигурации большинства пакетов под Linux. Ее стоит использовать в тех случаях, когда установка собранного пакета необязательно должна производиться в корень файловой системы.

В процессе конфигурации будет произведена проверка наличия необходимых зависимостей, сконфигурирован пакет с учетом всех условий текущей операционной системы и создан MakeFile. При выполнении данного пункта могут произойти различные ошибки, большинство из них связано с отсутствием необходимых пакетов.

После удачной конфигурации проекта стоит произвести сборку и установку пакета. Для этого необходимо последовательно выполнить команды:

```
make -j5  
make install
```

где опция `-j5` задает количество параллельных потоков при сборке, в данном случае количество потоков равно 5. Для расчета наиболее эффективного количества потоков применяется формула  $n+1$ , где  $n$  — это количество ядер процессора (либо физических, либо логических).

После этого в папке `/opt/crostool-ng` появится собранный комплект данной утилиты. Для удобства использования стоит в системную переменную `PATH` для текущего терминала добавить путь к исполняемому файлу утилиты.

```
export PATH=$PATH:/opt/crostool-ng/bin/
```

Так же, как и для сборки `crostool-ng` необходимо создать папку для сборки. Пусть данная папка будет иметь имя `build_toolchain`:

```
cd ..  
mkdir build_toolchain  
cd build_toolchain
```

Теперь необходимо запустить утилиту конфигурации и произвести настройку сборки кросскомпилятора под заданную платформу. Для запуска утилиты конфигурации надо выполнить следующую команду:

```
ct-ng menuconfig
```

В результате выполнения последней команды запустится утилита конфигурации с псевдографическим интерфейсом. В данном интерфейсе необходимо произвести настройки сборки под заданную архитектуру. Так же стоит отметить что в данном примере приводится конфигурация `crostool-ng-1.17.0`, поэтому в новых версиях могут быть различия.

#### Вкладка Paths & Misc:

Установить "Try features marked as EXPERIMENTAL"  
Задать "Prefix directory" ( `/opt/x-tools/${CT_TARGET}` )  
Pflnm "Number of parallel jobs" количества ядер на процессорщике

#### Вкладка Target options:

Задать "Target architecture" to "ARM"

Задать "Endianness" to "Little endian"  
Задать "Bitness" to "32-bit"  
Задать "Architecture level" to "armv6zk"  
Задать "Emit assembly for CPU" to "arm1176jzf-s"  
Задать "Tune for CPU" to "arm1176jzf-s"  
Задать "Use specific FPU" to "vfp"  
Задать "Floating point" to "hardware (FPU)"  
Задать "Default instruction set mode" to "arm"  
Установить "Use EABI"

Вкладка Toolchain options:

Задать "Tuple's vendor string" to "rpi"

Вкладка Operating system:

Задать "Target OS" to "linux"  
Задать "Linux kernel version" "3.6"  
Binary utilities:  
Set "Binary format" to "ELF"  
Set "binutils version" to "2.22"

Вкладка C compiler:

Установить "Show linaro versions"  
Задать "gcc version" to "linaro-4.7-2012.10"  
Установить "C++"  
Задать "gcc extra config" to "--with-float=hard"  
Установить "Link libstdc++ statically into gcc binary"

Вкладка C-library:

Задать "C library" to "eglibc"  
Задать "eglibc version" to "2\_13"

Более подробно значение каждого из пунктов утилиты конфигурации будет описано позже. После того как конфигурация произведена, необходимо сохранить файл конфигурации. И затем запустить непосредственно сборку кросскомпилятора. Для запуска сборки необходимо выполнить команду:  
ct-ng build

На данном этапе все ручные манипуляции. Далее пойдет автоматическое скачивание и сборка необходимых библиотек, модулей и т. д. По окончании сборки произойдет установка собранного кросскомпилятора в указанную в конфигурации папку. После чего можно будет использовать кросскомпилятор и сопутствующие утилиты.

Данный пример является упрощенным, существует ряд особенностей и нюансов. Кросскомпилятор, собранный данным методом подойдет для сборки простейших программ и пакетов. Для сборки более сложных

программ следует учитывать различные моменты, например, наличие в кросскомпиляторе дополнительных библиотек, скопированного rootfs целевого устройства особым образом, сочетание версий библиотек и компилятора установленного в системе устройства и самим кросскомпилятором и т. д.

### 3.2 Сборка crosstool-NG и кросскомпилятора для OrangePiZero

Данный пример сборки более объемный, но и более приближенный к боевым условиям.

#### Сборка crosstool-ng

Первым делом создаем папку в которой будем производить все манипуляции по сборке кросскомпилятора:

```
mkdir crosstools-ng
```

И переходим в нее:

```
cd /opt/buildOPIZ/crosstools-ng
```

Скачиваем архив с исходным кодом crosstool-ng:

```
wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-1.24.0.tar.bz2
```

И распаковываем его:

```
tar xvjf crosstool-ng-1.24.0.tar.bz2
```

Таким образом путь к исходному коду crosstool-ng будет выглядеть следующим образом:

```
/opt/buildOPIZ/crosstools-ng/crosstool-ng-1.24.0
```

Создаем папку для установки crosstool-ng:

```
mkdir crosstool-ng
```

Теперь необходимо установить все необходимые пакеты для сборки crosstool-ng:

```
sudo apt install libzip-dev
sudo apt-get install mercurial bison cvs gperf flex texinfo libtool automake build-essential libncurses-dev lzip device-tree-compiler subversion
```

Переходим в папку для сборки:

```
cd /opt/buildOPIZ/crosstools-ng/crosstool-ng-1.24.0
```

Производим сборку crosstool-ng:

```
./configure --prefix=/opt/buildOPIZ/crosstools-ng/crosstool-ng
make -j5
```

make install

Для ускорения процесса сборки в crosstool-ng можно создать репозиторий скаченных исходных кодов, он будет располагаться по следующему пути:

```
/opt/crosstool-ng_rep
```

### Настройка сборки кросскомпилятора

Создаем папку для сборки кросскомпилятора:

```
mkdir /opt/buildOPIZ/crosstools-ng/buildCrosscompile
```

И переходим в данную папку:

```
cd /opt/buildOPIZ/crosstools-ng/buildCrosscompile
```

Добавляем путь к ct\_ng:

```
export PATH="$PATH:/opt/buildOPIZ/crosstools-ng/crosstool-ng/bin"
```

Запускаем конфигурацию:

```
ct-ng menuconfig
```

Устанавливаем параметры сборки по таблице 2.1

Таблица 2.1 Парасетры сборки

Параметр	Значение
<i>Вкладка Paths and misc options</i>	
Local tarballs directory	/opt/crosstool-ng_rep
Save new tarballs	true
Prefix directory	/opt/x-tools/\${CT_HOST:+HOST- \${CT_HOST}/}\${CT_TARGET}
Render the toolchain read-only	false
Use a mirror	true
Number of parallel jobs	5
<i>Вкладка Target options</i>	
Target Architecture	arm
Default instruction set mode	arm
Endianness	Little endian
Bitness	32
Emit assembly for CPU	cortex-a7
Use specific FPU	vfpv4

Floating point

hard

Библиотека ГГТУ им. П.О.Сухого

<i>Вкладка Toolchain options</i>	
Tuple's vendor string	opiz
<i>Вкладка Operating System</i>	
Target OS	linux
Version of linux	4.15.18
<i>Вкладка Binary utilities</i>	
Version of binutils	2.30
Linkers to enable	ld, gold
Enable threaded gold	true
Add ld wrapper	true
Enable support for plugins	true
binutils libraries for the target	true
<i>Вкладка C-library</i>	
Version of glibc	2.23
Build libidn add-on	true
Create /etc/ld.so.conf file	true
<i>Вкладка C compiler</i>	
Version of gcc	6.5.0
gcc extra config	--with-float=hard
Compile libmudflap	true
Compile libgomp	true
Compile libssp	true
Compile libquadmath	true
Compile libsanitizer	true
C++	true
<i>Вкладка Debug facilities</i>	
gdb	true
<i>Вкладка Companion libraries</i>	
cloog	true
libelf	true
zlib	true
Build local zlib	true

<i>Вкладка Companion tools</i>	
Install companion tools for host	true
autoconf	true
automake	true
dtc	true
libtool	true
m4	true
make	true

После этого сохраняем конфигурацию и запускаем сборку:  
 ct-ngbuild

После удачного завершения сборки в указанном каталоге появится полный комплект для кроссплатформенной сборки. В исходном виде данный комплект подходит для сборки загрузчика и ядра. Для кросскомпиляции более объемных пакетов, которые имеют обширный круг зависимостей, необходимо править sysroot кросскомпилятора.

Для импортирования готового системного образа кросскомпилятора есть много разных способов, от простого копирования до синхронизации специальными утилитами. В случае обычного копирования возникает проблема исправления ссылок, т.к. путь к файлам в образе системного диска и в sysroot кросскомпилятора отличаются.

Для простого и успешного партирования системных файлов в кросскомпилятор лучше всего воспользоваться специализированной утилитой, внешний вид окна которой приведен на рисунке 2.12.

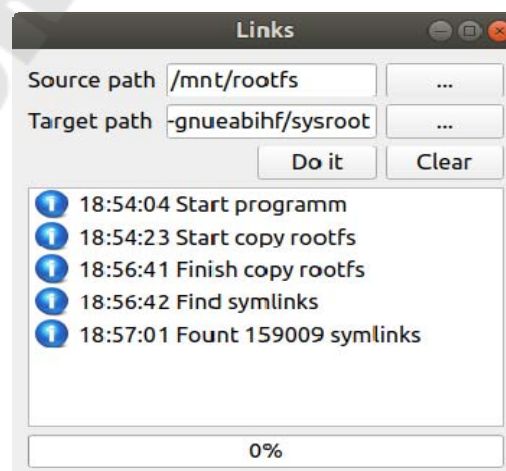


Рисунок 2.12 Внешний вид окна программы для партирования системных файлов.



### **3.3 Задание для самостоятельной работы**

Произвести сборку кросскомпилятора заданной версии с помощью пакета crosstools-ng под заданную платформу. Версию компилятора и платформу задает преподаватель индивидуально.

### **4. Содержание отчета**

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а так же снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Так же обязательно выполнить примеры из данной работы и вставить снимки экрана процесса выполнения примеров. Вывод о полученных навыках и знаний после выполнения работы.

### **Контрольные вопросы для самопроверки**

1. Что такое кросскомпиляция?
2. Что такое crosstools-ng?
3. Перечислите этапы сборки crosstools-ng и кросскомпилятора под заданную платформу?
4. Что такое sysroot кросскомпилятора?

## Лабораторная работа № 3

### Сборка загрузчика U-Boot под заданную платформу

#### 1. Цель работы

Изучить основные принципы сборки загрузчика U-Boot под заданную платформу.

#### 2. Основные теоретические сведения

Загрузчик Das U-Boot, наравне с известными lilo и grub, разработан для того, чтобы, запустившись после включения компьютера (будь то настольный PC или мобильный телефон), подготовить его к запуску основной операционной системы. Как и другие загрузчики, он облегчит выбор между вариантами загрузки, а также поможет при восстановлении после сбоя.

В отличии от своих "коллег", U-boot изначально использовался в системах на базе PowerPC, sparc, arm. Поддержка x86 появилась в нём сравнительно недавно.

##### 2.1 История разработки

Началом проекта является 8xx загрузчик PowerPC, называемый 8xxROM, написанный Магнусом Даммом. В октябре 1999 года Вольфганг Денка переместил проект на SourceForge.net и переименовал его в PPCBoot, потому что SF.net не позволял называть проект именами, начинающиеся с цифр. Версия 0.4.1 PPCBoot впервые была публично выпущена 19 июля 2000.

В 2002 году предыдущая версия исходного кода была на короткое время раздвоена в продукте под названием ARMBoot, но вскоре после этого была обратно объединена в проект PPCBoot.

В ноябре 2002 — был выпущен PPCBoot 2.0.0. Это был последний выпуск под названием PPCBoot, так как он был переименован, чтобы отразить поддержку ARM архитектуры в дополнение к PPC ISA.

PPCBoot-2.0.0 стал U-Boot-0.1.0 в ноябре 2002 г. с добавлением поддержки архитектуры x86. Дополнительная поддержка архитектуры была добавлена в последующие месяцы: MIPS32 — в марте 2003 года, MIPS64 — в апреле, Altera NIOS-32 — в октябре, Coldfire — в декабре, а Microblaze — в апреле 2004 года. В мае 2004 года выпуска U-Boot-1.1.2 включена поддержка 216 различных производителей плат с различными архитектурами.

В нынешнее название «Das U-Boot» добавлен немецкий определенный артикль, чтобы создать двуязычный каламбур с немецким словом «подводная лодка».

Это свободное программное обеспечение, распространяемое по условиям общественной лицензии GNU General. Оно может применяться на любой поддерживаемой архитектуре с использованием кросс-разработки

GNU инструментария, например, crosstool, Embedded Linux Development Kit (ELDK) или OSELAS.Toolchain.

Важность Das U-Boot в Embedded Linux систем достаточно лаконично изложена в книге Embedded Linux системы, Карим Ягмур. Текст о U-Boot начинается словами: «Хотя существует довольно много других загрузчиков, Das U-Boot, универсальный загрузчик, возможно, является самым богатым, самым гибким и наиболее активно развивающимся из загрузчиков с открытым исходным кодом».

## 2.2 Получение исходного кода

Проект U-Boot распространяется бесплатно в виде исходных кодов. Официальный сайт проекта <https://www.denx.de/wiki/U-Boot/WebHome>. Исходный код можно получить несколькими способами:

- 1) Получить текущую версию исходных кодов с помощью GIT с сервера проекта [git.denx.de](https://git.denx.de)
- 2) Получить стабильную версию с FTP сервера проекта <ftp://ftp.denx.de/pub/u-boot/>
- 3) Получить текущую версию с сервера GitHUB <https://github.com/u-boot/u-boot>

Таблица 3.1. Структура дерева исходного кода проектами

Папка	Описание
/arch	Специфические файлы поддержки архитектур
/arc	Файлы для архитектуры arc
/arm	Файлы для архитектуры arm
/m68k	Файлы для архитектуры m68k
/microblaze	Файлы для архитектуры microblaze
/mips	Файлы для архитектуры mips
/nds32	Файлы для архитектуры nds32
/nios2	Файлы для архитектуры nios2
/openrisc	Файлы для архитектуры openrisc
/powerpc	Файлы для архитектуры powerpc
/riscv	Файлы для архитектуры riscv
/sandbox	Файлы для архитектуры sandbox
/sh	Файлы для архитектуры sh
/x86	Файлы для архитектуры x86

/api	Машинно-независимый API для внешних приложений
/board	Файлы поддержки различных плат
/cmd	Файлы командой оболочки
/common	Разные архитектурно-независимые функции
/configs	Примеры различных конфигураций под различные устройства
/disk	Код для обработки раздела диска
/doc	Документация
/drivers	Часто используемые драйверы устройств
/dts	Содержит Makefile для сборки внутреннего U-Boot fdt
/examples	Пример кода для автономных приложений и т. д.
/fs	Драйвера поддержки файловых систем
/include	Заголовочные файлы
/lib	Библиотеки
/Licenses	Файлы лицензий
/net	Драйвера поддержки сети
/post	Тестирование при включении устройства (Power On Self Test)
/scripts	Различные скрипты для сборки
/test	Различные файлы модульных тестов
/tools	Инструменты для создания образов S-Record или U-Boot и т. д.

### 2.3 Особенности лицензирования

U-Boot является свободным программным обеспечением. Авторские права на него принадлежат Вольфгангу Денку и многим другим, кто предоставил код. U-Boot можно распространять U-Boot и / или изменять его в соответствии с условиями 2 версии Стандартной общественной лицензии GNU, опубликованной Free Software Foundation. Большая часть его также может распространяться, по выбору, под любой более поздней версией Стандартной общественной лицензии GNU.

## 2.4 Системная утилита dd

Системная утилита `dd` предназначена для побайтового копирования информации. Общая структура команды для использования данной утилиты выглядит следующим образом:

```
dd [--help] [--version] [if=файл] [of=файл] [ibs=байты] [obs=байты]
[bs=байты][cbs=байты] [skip=блоки] [seek=блоки] [count=блоки]
[conv={ascii, ebclic, ibm, block, unblock, lcase, ucase, swab, noerror, notrunc,
sync}]
```

Утилита `dd` копирует файл (по умолчанию из стандартного ввода на стандартный вывод), используя заданные размеры блоков для ввода и вывода, и в то же время, возможно, выполняя его преобразование.

В таблице 3.2 приведен перечень основных параметров утилиты `dd`.

Таблица 3.2. Основные параметры утилиты `dd`

Опция	Описание
<code>if=файл</code>	Читает данные из <i>файла</i> вместо стандартного ввода.
<code>of=файл</code>	Пишет данные в <i>файл</i> вместо стандартного вывода. Если только не задан <code>conv=notrunc</code> , <code>dd</code> обрезает <i>файл</i> до нулевого размера (или размера, заданного <code>vseek=</code> ).
<code>ibs=bytes</code>	Читает по <i>bytes</i> байт за раз. По умолчанию 512.
<code>obs=bytes</code>	Пишет по <i>bytes</i> байт за раз. По умолчанию 512.
<code>bs=bytes</code>	Читает и пишет по <i>bytes</i> байт за раз. Данная опция перекрывает опции <code>ibs</code> и <code>obs</code> . (Кроме того, установка <code>bs</code> не эквивалентна установке обеих опций <code>ibs</code> и <code>obs</code> в то же значение, по крайней мере, когда не задано преобразований отличных от <code>sync</code> , <code>noerror</code> и <code>notrunc</code> , так как она оговаривает, что каждый входной блок будет копироваться на выход как отдельный блок без объединения коротких блоков).
<code>cbs=байт</code>	Задаёт размер блока для преобразований типа <code>block</code> и <code>unblock</code> .
<code>skip=block</code>	Пропускает <i>blocks</i> блоков длины <code>ibs</code> байт во входном файле перед началом копирования.
<code>seek=blocks</code>	Пропускает в выходном файле <i>blocks</i> блоков длины <code>obs</code> байт перед началом копирования.
<code>count=blocks</code>	Копирует лишь <i>blocks</i> блоков длины <code>ibs</code> байт из входного файла, а не весь входной файл, как обычно.
<code>conv=ПРЕОБРАЗОВАНИЕ</code>	Преобразует файл, как задано аргументом(ами) <i>ПРЕОБРАЗОВАНИЕ</i> . (Вокруг запятых не должно быть пробелов).

Таблица 3.3 Типы преобразований

Преобразование	Описание
ascii	Преобразование EBCDIC в ASCII.
ebcdic	Преобразование ASCII в EBCDIC.
ibm	Преобразование ASCII в альтернативный EBCDIC.
block	Для каждой строки во входном файле, выводить <b>chs</b> байт, заменяя символ новой строки на пробел и добывая пробелами при необходимости.
unblock	Заменять заключительные пробелы в каждом входном блоке размера <b>chs</b> байт на символ новой строки.
lcase	Изменять заглавные буквы на строчные.
ucase	Изменять строчные буквы на заглавные.
swab	Менять местами каждую пару входных байт. Если последний байт имеет нечетный порядковый номер, то он просто копируется (так как менять местами нечего). [POSIX 1003.2b, PASC interpretations 1003.2 #3 и #4]
noerror	Продолжать после ошибок чтения.
notrunc	Не обрезать выходной файл.
sync	Дополнять каждый входной блок до размера <b>ibs</b> путем добавления нулевых байт.

## 2.5 Системная утилита losetup

Системная утилита losetup предназначена для ассоциации файла-образа с блочным устройством. В операционных системах семейства Linux к блочным устройствам относятся все диски (жесткие диски, флеш-карты, SSD накопители и т. д.).

Пример использования утилиты losetup:

Вывести информацию:

```
losetup loopdev
    losetup -l [-a]
    losetup -j file [-o offset]
```

Отключить loop устройство:

```
losetup -d loopdev...
```

Отключить все loop устройства:

```
losetup -D
```

Вывести имя первого свободного loop устройства:

```
losetup -f
```

Установить loop устройство:

```
losetup [-o offset] [--sizelimit size]
```

`[-Pr] [--show] -f|loopdev file`

Изменить размер loop устройства:

`losetup -c loopdev`

В таблице 3.4 приведен перечень основных параметров утилиты `losetup`.

Таблица 3.4 Основные параметры утилиты `losetup`.

Опция	Описание
<code>-a, --all</code>	Показывает статус всех loop устройств. Не все данные доступны для пользователей без полномочий <code>root</code> .
<code>-c, --set-capacity loopdev</code>	Перечитать размер файла, связанного с указанным loop устройством.
<code>-d, --detach loopdev...</code>	Отсоединение файл или устройства, связанного с указанным loop устройством.
<code>-D, --detach-all</code>	Отсоединить все loop устройства.
<code>--direct-io[=on off]</code>	Включение или отключение прямого ввода-вывода для файла резервной копии. Необязательный аргумент может быть включен или выключен. Если аргумент опущен, он по умолчанию включен.
<code>-f, --find</code>	Поиск первого свободного loop устройства. Если указано имя файла, то найденное loop устройство будет использовано для подключения указанного файла. Если имя файла отсутствует то будет выведено имя найденного устройства.
<code>-L, --nooverlap</code>	Проверить конфликты между loop устройствами, чтобы избежать ситуации, когда один и тот же файл используется совместно с другими loop устройствами. Если файл уже используется другим устройством, то не создавать новую ассоциацию, а использовать существующую. Опция имеет смысл только с <code>--find</code> .
<code>-j, --associated file</code>	Показать статус всех loop устройств, связанных с данным файлом.
<code>-J, --json</code>	Использовать JSON формат для вывода <code>--list</code> .
<code>-l, --list</code>	Если указано loop устройство или параметр <code>-a</code> , выводятся столбцы по умолчанию для указанного loop устройства или всех устройств; по умолчанию используется для печати информации обо всех устройствах.
<code>-n, --noheadings</code>	Не печатать заголовки для выходного формата.

-o, --offset offset	The data start is moved offset bytes into the specified file or device.
-O, --output columns	Спецификация столбцов информации для вывода --list.
-P, --partscan	Заставить ядро сканировать таблицу разделов на вновь созданное устройство цикла.
--raw	Использовать выходной формат raw для --list.
-r, --read-only	Ассоциировать как устройство только для чтения.
--show	Показать имя loop устройства ассоциированного с файлом.
-v, --verbose	Подробный режим

### 3. Порядок выполнения работы

Сначала стоит рассмотреть процесс сборки загрузчика для одноплатного ПК Orange Pi Zero.

#### 3.1 Создание образа загрузочного диска для Orange Pi Zero

Основой для всей операционной системы одноплатного компьютера является образ загрузочного диска (флеш-карты). Непосредственно на нем располагается загрузчик, раздел с конфигурацией загрузчика и ядром операционной системы, а так же раздел содержащий корневую файловую систему. Создание основы для образа загрузочного диска в общем случае состоит из следующих этапов:

1. Создание файла, размер которого равен предполагаемому размеру системного диска.
2. Ассоциация данного файла с блочным устройством.
3. Создание таблицы разделов.
4. Создание раздела для хранения конфигурации загрузчика и ядра операционной системы.
5. Создание раздела для хранения корня файловой системы.

Сначала необходимо создать файл, размер которого будет приблизительно таким же, как и размер системного диска. Для этого необходимо произвести копирование необходимого количества байт из устройства /dev/zero в файл образа. Устройство /dev/zero выполняет только одну функцию: каждый раз, когда из него производится считывание, оно возвращает запрошенное количество байт, каждый из которых равен 0. Для побайтового копирования информации применяется системная утилита dd. Следующая команда создает файл размером 6Гб:

```
sudo dd if=/dev/zero of=test.img bs=1M count=6000
```



где `bs` - размер буфера, записываемого за один раз, `count` - количество циклов записи, `if` - источник данных, `of` — место записи данных. Таким образом будет создан пустой файл размером 6Гб.

Далее необходимо создать таблицу разделов, а так же файловые системы, однако данные действия относятся только к дискам. Для того, чтобы была возможность работать с файлом так же, как и с диском необходимо ассоциировать его с блочным устройством. Данное действие можно произвести с помощью системной утилиты `losetup`. Следующая команда выполняет ассоциацию файла с одним из свободных блочных устройств, производит сканирование данного файла на наличие таблицы разделов:

```
sudo losetup --partscan --show --find test.img
```

После выполнения команды должна появиться надпись типа `«/dev/loop1»` - это имя блочного устройства, которое ассоциировано с нашим файлом. По сути теперь наш файл стал диском, и с ним можно работать любыми дисковыми утилитами (например `gparted` или `fdisk`).

Следующим шагом необходимо создать таблицу разделов на данном блочном устройстве, а так же создать необходимые разделы. Следующая команда создает таблицу разделов, и два раздела: первый размером 100Мб, на котором будут располагаться конфигурация загрузчика и ядро операционной системы и второй, который займет все свободное пространство и на котором будет располагаться корневая файловая система:

```
sudo blockdev --rereadpt /dev/loop1
```

```
sudo -i
```

```
export blockDev="/dev/loop1"
```

```
cat <<EOT | sfdisk /dev/loop1
```

```
1M,100M,c
```

```
„L
```

```
EOT
```

```
exit
```

Здесь `«start»` - это номер 1к-блока. (Умножьте его на два, чтобы получить соответствующий номер сектора - при условии 512-байтных секторов). В таблице 3.5 приведен пример распределения места на системном носителе.

Таблица 3.5 Пример распределения места на системном диске

Начало	Размер	Предназначение
0	8KB	Не используется, доступно для таблицы разделов и т. д.
8	24KB	Начальный загрузчик SPL
32	512KB	U-Boot
544	128KB	Окружающая среда(environment)
672	352KB	Зарезервировано
2048	-	Свободно для разделов

После создания таблицы разделов необходимо произвести повторное чтение диска, для этого выполняется следующая команда

```
sudo blockdev --rereadpt /dev/loop1
```

Для внесения данных о разделах выполняем следующую команду:

```
sudo sfdisk /dev/loop1
```

Затем заносим данные разделов. Для первого раздела вводим 1М,100М,с. Для второго раздела вводим,L.Затем нажимаем CTRL+D и нажимаем Y для подтверждения записи изменений на диск.

После создания разделов нужно создать файловые системы для каждого. Первый раздел имеет небольшой объем и будет использоваться загрузчиком, поэтому файловая его файловая система должна быть простой и легковесной. В связи с этим для первого раздела выбирается файловая система FAT16. Второй раздел имеет значительный объем, а так же предназначен для хранения большого количества файлов и должен поддерживать отдельные права доступа. Поэтому для второго раздела выбирается более мощная файловая система EXT4. Следующая последовательность команд создаст обе файловые системы:

```
sudo mkfs.vfat /dev/loop1p1
```

```
sudo mkfs.ext4 /dev/loop1p2
```

На первом разделе будет создана файловая система fat16. На втором разделе будет создана файловая система ext4.

Далее можно производить копирование необходимых файлов, запись загрузчика и т. д. Для возможности работы с диском его необходимо смонтировать. Для этого сначала создается каталог для монтирования каждого диска:

```
sudo mkdir /mnt/arm1
```

```
sudo mkdir /mnt/arm2
```

Для монтирования разделов необходимо выполнить следующие команды:

```
sudo mount /dev/loop1p1 /mnt/arm1
```

```
sudo mount /dev/loop1p2 /mnt/arm2
```

Теперь при произведении действий в данных папках (создание папок, файлов, запись данных) вся информация будет сохраняться в файл образа test.img.

Для размонтирования разделов применяется команда umount. Следующая последовательность команд позволяет размонтировать разделы диска:

```
sudo umount /dev/loop1p1
```

```
sudo umount /dev/loop1p2
```

После выполнения этих команд в папках /mnt/arm1 и /mnt/arm2 уже не будут доступны разделы файла образа.

После завершения всех необходимых действий, ассоциация образа с блочным устройством должна быть снята (не стоит забывать, что перед этим действием все файловые системы должны быть размонтированы). Для снятия ассоциации выполняется следующая команда:

```
sudo losetup -d /dev/ loop1
```

Для записи образа системного диска на флеш-карту можно воспользоваться утилитой `dd`, которая в побайтовом режиме полностью запишет образ системного диска на карту:

```
dd if= test.img of=/dev/sdb bs=1024
```

где `/dev/sdb` целевая SD карта

### 3.2 Сборка U-Boot для Orange Pi Zero

Рассмотри пример сборки загрузчика для Orange Pi Zero. Хотя загрузчик и более зависим от аппаратной платформы, чем остальные составляющие части системы, но процесс сборки в общем виде одинаковый для различных платформ. Для сборки загрузчика потребуется кросскомпилятор, сборка которого была рассмотрена в предыдущей лабораторной работе.

Для начала необходимо создать каталог, в котором будут производиться все манипуляции по сборке:

```
mkdir /opt/buildOPIZ/u-boot
```

А так же необходимо перейти в него:

```
cd /opt/buildOPIZ/u-boot
```

Для сборки загрузчика так же необходимо установить зависимости. Зависимости — это те программные пакеты, без которых процесс сборки не произойдет. В случае U-Boot потребуются программные пакеты: `device-tree-compiler` (программный пакет для сборки файлов дерева устройств), `u-boot-tools` (набор утилит `u-boot`) и `swig` (свободный инструмент для связывания программ и библиотек, написанных на языках C и C++, с интерпретируемыми). Для установки данных пакетов необходимо ввести следующую последовательность команд:

```
sudo apt install device-tree-compiler
```

```
sudo apt install u-boot-tools
```

```
sudo apt-get install swig
```

Исходный код данного загрузчика распространяется по лицензии GPL, в следствии чего получить его исходный не составит проблем. На официальном сервере проекта располагается репозиторий GIT, в котором и ведется разработка. Для получения исходного кода его необходимо сканировать:

```
git clone git://git.denx.de/u-boot.git
```

После клонирования в текущем каталоге появится папка под названием `u-boot`. Необходимо перейти в нее:

```
cd /opt/buildOPIZ/u-boot/u-boot
```

В случае, если с данной копией исходного кода уже проводилась сборка загрузчика, то стоит ее очистить, иначе могут возникнуть проблемы при повторной сборке. Для этого необходимо в папке с исходным кодом выполнить две команды:

```
make clean
```

```
make mrproper
```

Как и большинство подобных проектов, U-Boot имеет обширный список примеров конфигурации под различные устройства. Для получения данного списка необходимо выполнить следующие две команды:

```
tools/genboardscfg.py  
grep sunxi boards.cfg | awk '{print $7}'
```

В результате их выполнения, на экран будет выведен список готовых примеров конфигурации. Если конфигурация под заданное устройство присутствует в данном списке, то можно ее использовать для начальной конфигурации загрузчика. В данном случае необходима конфигурация с именем `orangepi_zero_defconfig`.

В текущей версии загрузчика для процессоров из семейства `sun8i` не установлена поддержка ядра операционной системы в сжатом виде. Для того чтобы данная поддержка присутствовала в загрузчике необходимо внести изменения в файл `include/configs/sun8i.h`:

```
nano include/configs/sun8i.h открываем файл
```

```
вструктуру  
#ifndef __CONFIG_H  
#define __CONFIG_H  
  
...  
  
#endif /* __CONFIG_H */
```

```
добавим строку  
#define CONFIG_CMD_BOOTZ
```

После внесения изменений в исходный код можно произвести начальную конфигурацию загрузчика конфигурацией по умолчанию, которая была выбрана ранее. Для этого нужно запустить утилиту `make`, указав ей полный путь к кросскомпилятору и файл конфигурации:

```
make CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/f/bin/arm-  
opiz-linux-gnueabi/f- orangepi_zero_defconfig
```

После данной операции, в папке с исходным кодом появится файл `.config` в котором будет содержаться конфигурация для сборки. Если конфигурации по умолчанию не хватает, то можно произвести дополнительные настройки через встроенную утилиту с псевдографическим интерфейсом. Для запуска данной утилиты так же необходимо вызвать утилиту `make` указав её полный путь к кросскомпилятору и дополнительную опцию `menuconfig`:

```
make -j5 CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-  
gnueabi/f/bin/arm-opiz-linux-gnueabi/f- menuconfig
```

После того как все необходимые настройки выполнены, можно запустить процесс сборки. Для этого снова надо вызвать утилиту `make`, указав ей количество потоков сборки, а так же полный путь к кросскомпилятору:

```
make -j5 CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi-
```

При удачной сборке должен появиться файл `u-boot-sunxi-with-spl.bin`, который собственно и является загрузчиком. Для того, чтобы процессор смог его выполнить необходимо его записать в загрузочную область системного диска. В предыдущем разделе был описан процесс создания образа системного диска. Необходимо взять файл образа и ассоциировать его с блочным устройством. Затем необходимо произвести запись собранного загрузчика непосредственно на образ системного диска. Для записи необходимо воспользоваться утилитой `dd`, указав файл с кодом загрузчика и блочное устройство, с которым ассоциирован файл образа:

```
sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/loop2 bs=1024 seek=8
```

После записи кода загрузчика необходимо произвести настройки. Все настройки загрузчика хранятся на первом разделе образа системного диска. Сначала необходимо его смонтировать в подготовленную папку:

```
sudo mount /dev/loop2p1 /mnt/boot
```

На первом разделе необходимо создать конфигурацию для работы загрузчика. Для создания `boot.scr` из файла `boot.cmd` необходима утилита `mkimage` из пакета `u-boot-tools`. После монтирования раздела необходимо создать исходный файл настроек `boot.cmd`:

```
sudo nano /mnt/boot/boot.cmd
```

Данный файл должен содержать конфигурацию, которая будет определять алгоритм работы загрузчика. В данном случае прописывается имя файла ядра, имя файла дерева устройств, а так же дополнительные параметры конфигурации. К таким параметрам можно отнести порт, в который будет выводиться системный лог, скорость работы данного порта, расположение системного раздела и т.д. Пример такой конфигурации приведен ниже.

```
fatload mmc 0 0x46000000 zImage
```

```
fatload mmc 0 0x49000000 sun8i-h2-plus-orangepi-zero.dtb
```

```
setenv bootargs console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p2 rw  
rootwait panic=10
```

```
bootz 0x46000000 - 0x49000000
```

После создания исходного файла конфигурации, необходимо создать скомпилированный файл конфигурации. Для этого используется утилита `mkimage`:

```
sudo mkimage -C none -A arm -T script -d /mnt/boot/boot.cmd  
/mnt/boot/boot.scr
```

После этого процесс сборки и установки загрузчика можно считать завершенным. После всех манипуляций не стоит забывать размонтировать все разделы и отменить ассоциацию файла образа с блочным устройством.

### **3.3 Задание для самостоятельного выполнения**

Произвести создание образа системного диска, сборку и установку загрузчика U-Boot с использованием кросскомпилятора под заданную платформу. Версию загрузчика и платформу задает преподаватель.

### **4. Содержание отчета**

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а так же снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Так же обязательно выполнить примеры из данной работы и вставить снимки экрана процесса выполнения примеров. Вывод о полученных навыках и знаниях после выполнения работы.

### **Контрольные вопросы для самопроверки**

1. Для чего предназначена системная утилита dd?
2. Для чего предназначена системная утилита losetup?
3. Как производится ассоциация образа диска с блочным устройством?
4. Что такое U-Boot и для чего он предназначен?
5. Каким образом производится начальная конфигурация загрузчика?
6. Как производится сборка загрузчика?
7. Как производится установка загрузчика?

## Лабораторная работа № 4

### Сборка ядра и модулей по заданную платформу

#### 1. Цель работы

Изучить основные принципы сборки ядра ОС Linux и модулей по заданную платформу.

#### 2. Основные теоретические сведения

##### 2.1 Краткие сведения о ядре Linux

Операционная система Linux была разработана Линусом Торвальдсом (Linus Torvalds) в 1991 году как операционная система для компьютеров, работающих на новом и самом передовом в то время микропроцессоре Intel 80386. Тогда Линус Торвальдс был студентом университета в Хельсинки и был крайне возмущен отсутствием мощной и в то же время свободно доступной Unix-подобной операционной системы. Правившая бал в то время операционная система DOS, продукт корпорации Microsoft, была для Торвальдса полезна только лишь для того, чтобы поиграть в игрушку “Принц Персии”, и не для чего больше. Линус пользовался операционной системой Minix, недорогой Unix-подобной операционной системой, которая была создана в качестве учебного пособия. В этой операционной системе ему не нравилось отсутствие возможности легко вносить и распространять изменения в исходном коде (это запрещалось лицензией ОС Minix), а также технические решения, которые использовал автор ОС Minix.

Из-за неуклонного роста возможностей и не очень качественного построения некоторых современных операционных систем понятие операционная система стало несколько размытым. Многие пользователи считают, что то, что они видят на экране, — и есть операционная система. В это понятие входят ядро и драйверы устройств, системный загрузчик (boot loader), командная оболочка или любой другой интерфейс пользователя, а также базовая файловая система и системные утилиты.

В своей основе ядро операционной системы — это программное обеспечение, которое предоставляет базовые функции для всех остальных частей операционной системы, управляет аппаратным обеспечением и распределяет системные ресурсы.

Ядро часто называют супервизором (supervisor), основной частью (core) или внутренностями (internals) операционной системы. Типичные компоненты ядра – обработчики прерываний, которые обслуживают запросы на прерывания, поступающие от различных устройств, планировщик, распределяющий процессорное время между многими процессами, система управления памятью, которая управляет адресным пространством процессов, и системные службы, такие как сетевая подсистема и подсистема

межпроцессного взаимодействия. В современных системах с устройствами управления защищенной памятью ядро обычно занимает привилегированное положение по отношению к пользовательским программам. Это включает доступ ко всем областям защищенной памяти и полный доступ к аппаратному обеспечению. Системные переменные (system state) и область памяти, в которой находится ядро, вместе называются пространством ядра (kernel-space), или привилегированным режимом. Соответственно, пользовательские программы выполняются в пространствах задач (user-space), или в пользовательском режиме. Пользовательским программам доступно лишь некоторое подмножество машинных ресурсов, они не могут выполнять некоторые системные функции, напрямую работать с аппаратурой, обращаться к системной памяти (за пределами адресного пространства, выделенного пользовательской программе ядром) и делать другие недозволённые вещи. При выполнении программного кода ядра система переходит в пространство ядра, или переключается в привилегированный режим. При запуске обычных процессов система переключается в пользовательский, или непривилегированный режим.

Прикладные программы, работающие в системе, взаимодействуют с ядром с помощью системных вызовов {system call). Прикладная программа обычно вызывает функции различных библиотек, например, библиотеки функций языка С, которые в свою очередь обращаются к системным вызовам для того, чтобы отдать приказ ядру выполнить определенные действия от их имени. Некоторые библиотечные вызовы выполняют функции, для которых отсутствует системный вызов, и поэтому обращение к ядру — это только один этап в более сложной функции (рис. 4.1).

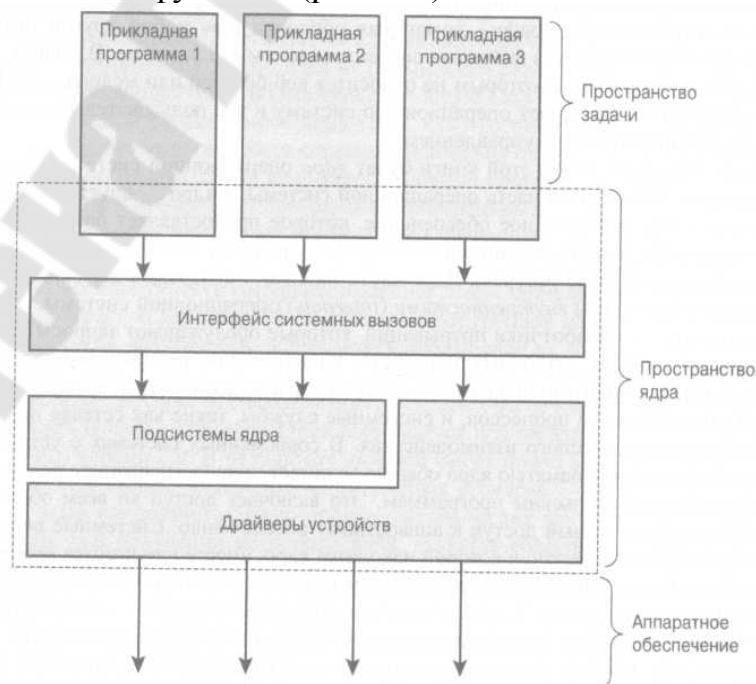


Рисунок 4.1. Взаимодействие прикладных программ, ядра и аппаратного обеспечения

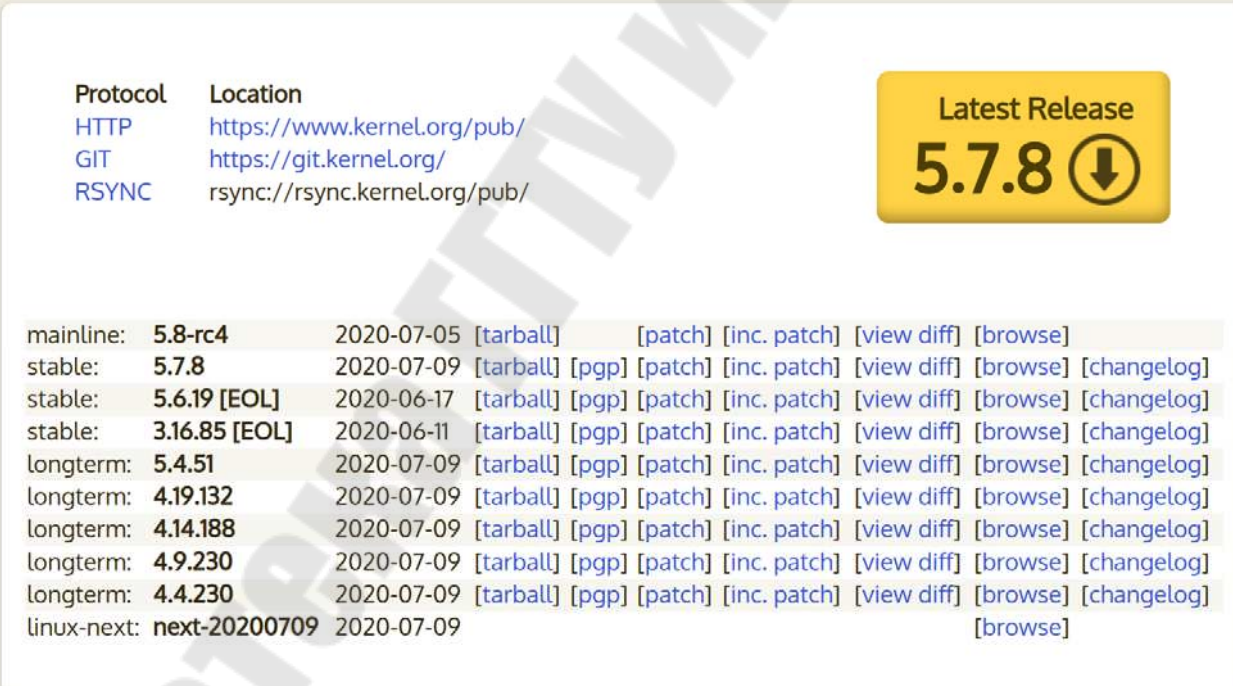


## 2.2 Получение исходного кода ядра

Самое первое что нужно сделать - это скачать исходники ядра. Исходники лучше брать с сайта конкретного дистрибутива, если они там есть или официального сайта ядра: [kernel.org](https://kernel.org). Рассмотрим загрузку исходников с [kernel.org](https://kernel.org).

Перед тем как скачивать исходники нам нужно определиться с версией ядра которую необходимо собирать. Есть две основных версии релизов - стабильные (stable) и кандидаты в релизы (rc), есть, конечно, еще стабильные с длительным периодом поддержки (longterm) но важно сейчас разобраться с первыми двумя. Стабильные это, как правило, не самые новые, но зато уже хорошо протестированные ядра с минимальным количеством багов. Тестовые - наоборот, самые новые, но могут содержать различные ошибки.

После того как необходимая версия ядра выбрана нужно перейти на сайт [kernel.org](https://kernel.org) и скачать архив `tar.xz` с необходимой версией исходного кода ядра Linux (рис. 4.2).



The screenshot shows the Linux kernel website interface. At the top left, there are download options for different protocols:

Protocol	Location
HTTP	<a href="https://www.kernel.org/pub/">https://www.kernel.org/pub/</a>
GIT	<a href="https://git.kernel.org/">https://git.kernel.org/</a>
RSYNC	<a href="rsync://rsync.kernel.org/pub/">rsync://rsync.kernel.org/pub/</a>

At the top right, there is a yellow button labeled "Latest Release" with the version number "5.7.8" and a download icon.

Below this, there is a list of kernel versions with their release dates and links to download or view details:

Version	Date	tarball	pdp	patch	inc. patch	view diff	browse	changelog
mainline: 5.8-rc4	2020-07-05	[tarball]		[patch]	[inc. patch]	[view diff]	[browse]	
stable: 5.7.8	2020-07-09	[tarball]	[pdp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
stable: 5.6.19 [EOL]	2020-06-17	[tarball]	[pdp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
stable: 3.16.85 [EOL]	2020-06-11	[tarball]	[pdp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm: 5.4.51	2020-07-09	[tarball]	[pdp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm: 4.19.132	2020-07-09	[tarball]	[pdp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm: 4.14.188	2020-07-09	[tarball]	[pdp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm: 4.9.230	2020-07-09	[tarball]	[pdp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm: 4.4.230	2020-07-09	[tarball]	[pdp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
linux-next: next-20200709	2020-07-09						[browse]	

Рисунок 4.2. Внешний вид сайта с различными версиями ядра Linux

Так же можно получить самую свежую текущую версию из репозитория GIT с помощью команды `git clone https://github.com/torvalds/linux`.

## 2.3 Установка зависимостей и выбор варианта конфигурации

В общем случае для сборки ядра потребуются следующие пакеты:

- build-dep
- linux
- kernel-package

Установить их можно с помощью следующей команды:

```
sudo apt-get update
sudo apt-get build-dep linux
sudo apt-get install kernel-package
```

Конфигурация сборки ядра может происходить различными путями. Каждый разработчик использует наиболее удобный для него способ. Список доступных вариантов выглядит следующим образом:

- `config` - традиционный способ конфигурирования. Программа выводит параметры конфигурации по одному, предлагая вам установить для каждого из них свое значение. Не рекомендуется для неопытных пользователей.
- `oldconfig` - файл конфигурации создается автоматически, основываясь на текущей конфигурации ядра. Рекомендуется для начинающих.
- `defconfig` - файл конфигурации создается автоматически, основываясь на значениях по умолчанию.
- `menuconfig` - псевдографический интерфейс ручной конфигурации, не требует последовательного ввода значений параметров. Рекомендуется для использования в терминале.
- `xconfig` - графический (X) интерфейс ручной конфигурации, не требует последовательного ввода значений параметров.
- `gconfig` - графический (GTK+) интерфейс ручной конфигурации, не требует последовательного ввода значений параметров. Рекомендуется для использования в среде GNOME.
- `localmodconfig` - файл конфигурации, создающийся автоматически, в который включается только то, что нужно данному конкретному устройству. При вызове данной команды большая часть ядра будет замодулирована.
- `localyesconfig` - файл конфигурации, похожий на предыдущий, но здесь большая часть будет включена непосредственно в ядро. Идеальный вариант для начинающих.

В случае, если вы хотите использовать `config`, `oldconfig`, `defconfig`, `localmodconfig` или `localyesconfig`, вам больше не нужны никакие дополнительные пакеты. В случае же с оставшимися тремя вариантами необходимо установить также дополнительные пакеты.

При использовании `menuconfig` потребуется пакет `libncurses5-dev`. Которые отвечает за построение псевдографического интерфейса. При использовании `gconfig` потребуются пакеты `libgtk2.0-dev` `libglib2.0-dev` `libglade2-dev`, которые отвечают за построение графического интерфейса на основе GTK.

При использовании `xconfig` потребуется библиотека `libqt4-dev`, которая отвечает за построение графического интерфейса на основе Qt. Либо понадобится прописать в системе полный путь к вашему комплекту Qt.

### 3. Порядок выполнения работы

#### 3.1 Сборка ядра Linux для Orange Pi Zero

Рассмотрим пример сборки ядра для операционной системы на платформе Orange Pi Zero с добавлением дополнительных модулей. Устройство на одноплатном компьютере в данном примере будет состоять из следующих компонентов:

- Одноплатный ПК Orange Pi Zero
- Комбинированный модуль GSM + GPS Neoway N720.
- USB флеш-накопитель.
- USB-UART преобразователи.

Как и в предыдущей работе, в первую очередь необходимо создать каталог, в котором будут производиться все манипуляции по сборке с исходным кодом. Для создания каталога выполним следующую команду:

```
mkdir /opt/buildOPIZ/kernel
```

А так же сразу переходим в него:

```
cd /opt/buildOPIZ/kernel
```

Исходный код ядра имеет лицензию GPL, в следствии чего он распространяется бесплатно в виде исходных кодов. Существует много различных путей, как его получить, однако основных два метода – скачать архив с интересующей версией с официального сайта `kernel.org` либо клонировать репозиторий GIT. В следствии того, что ядро Linux довольно старый проект, репозиторий гит содержит в себе очень много информации и занимает большой объем. Поэтому, если нет особой необходимости, исходный код лучше всего получить в виде архива, который содержит необходимую версию ядра. В данном примере будет применяться ядро версии 4.15. Для его получения необходимо выполнить следующую команду:

```
wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.15.18.tar.gz
```

В результате выполнения данной команды произойдет скачивание указанного файла в текущую папку. Т.к. В данном случае исходный код ядра содержится в архиве и поэтому перед проведением манипуляций по сборке

необходимо его распаковать. Следующая команды распаковывает архив с исходным кодом ядра:

```
tarxvzflinux-4.15.18.tar.gz
```

После распаковки архива, в текущей папке появится каталог с исходным кодом ядра Linux. Для продолжения сборки стоит перейти в него:

```
cd /opt/buildOPIZ/kernel/linux-4.15.18
```

Как и в случае с загрузчиком, если с данным исходным кодом уже производилась сборка, то стоит очистить исходный код от файлов предыдущей сборки. Это можно сделать выполнив следующую последовательность команд:

```
makeclean
```

```
makemrproper
```

Вместе с исходным кодом ядра поставляется и набор конфигураций под различные платформы. Список конфигураций для архитектуры ARM можно увидеть в каталоге arch/arm/configs. Если в списке присутствует конфигурация для заданной платформы, то необходимо выполнить начальную конфигурацию ядра. В данном случае конфигурация для процессоров производства фирмы AllWinner является sunxi\_defconfig. Для выполнения конфигурации ядра необходимо вызвать утилиту make, указав ей архитектуру, полный путь к кросскомпилятору и имя файла конфигурации.

```
make ARCH=arm CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi- sunxi_defconfig
```

В случае если необходима поддержка дополнительных устройств, необходимо выполнить дополнительную конфигурацию ядра. В данном случае устройство должно поддерживать 4 типа дополнительных устройств.

Комбинированный модуль GSM + GPSNeowayN720 не имеет специализированного драйвера, однако есть возможность включения его поддержки в стандартном драйвере. Для этого необходимо открыть файл drivers/usb/serial/option.c и добавить в него VID и PID данного устройства (рис. 4.3).

Открытие файла производится с помощью следующей команды:

```
nano drivers/usb/serial/option.c
```

Далее необходимо добавить строку с VID устройства:

```
#define NEOWAY_N720_VENDOR_ID 0x2949
```

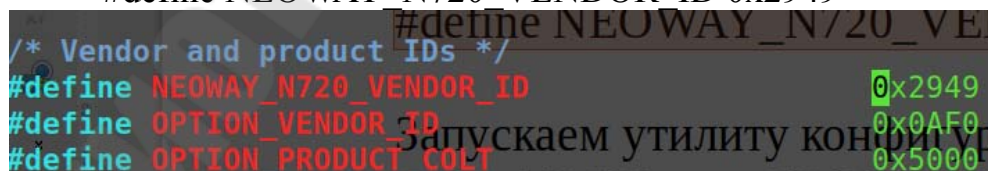


Рисунок 4.3. Добавление VID устройства

Затем в структуру `structusb_device_id` необходимо добавить PID устройства (рис. 4.4). Модуль N720 может иметь два различных идентификатора в зависимости от режима работы, потому для универсальность добавляем оба в данную структуру:

```
{ USB_DEVICE(NEOWAY_N720_VENDOR_ID,0x8241) },
```

```

        { USB_DEVICE(NEOWAY_N720_VENDOR_ID,0x8242) },
static const struct usb_device_id option_ids[] = {
    { USB_DEVICE(NEOWAY_N720_VENDOR_ID,0x8241) },
    { USB_DEVICE(NEOWAY_N720_VENDOR_ID,0x8242) },
    { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_COLT) },

```

Рисунок 4.4. Добавление PID устройства

Для настройки сборки встроенных устройств можно запустить специальную утилиту конфигурации. Существует два варианта утилиты конфигурации: консольная утилита с псевдографическим интерфейсом и утилита с полноценным графическим интерфейсом (рис. 4.5).

Для запуска консольной утилиты конфигурации необходимо запустить утилиту make, передав ей архитектуру, полный путь к кросскомпилятору и опцию menuconfig:

```
make ARCH=arm CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi- menuconfig
```

Для запуска утилиты конфигурации с полноценным графическим интерфейсом, необходимо выполнить соответствующую команду, а так же предварительно установить из репозитория библиотеки Qt. Для запуска утилиты конфигурации необходимо запустить утилиту make, передав ей архитектуру процессора, полный путь к конфигурации, а так же опцию xconfig.

```
make ARCH=arm CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi- xconfig
```

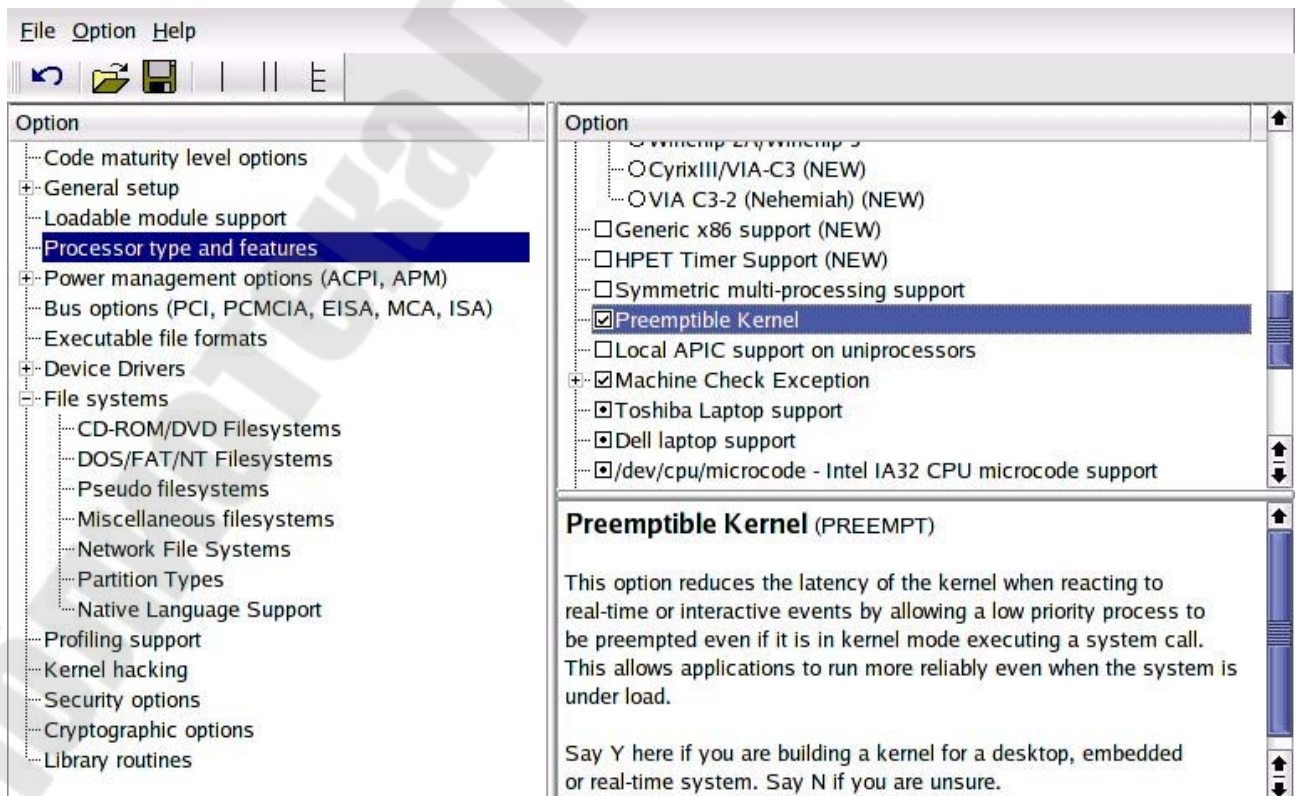


Рисунок 4.5. Внешний вид утилиты конфигурации ядра с графическим интерфейсом

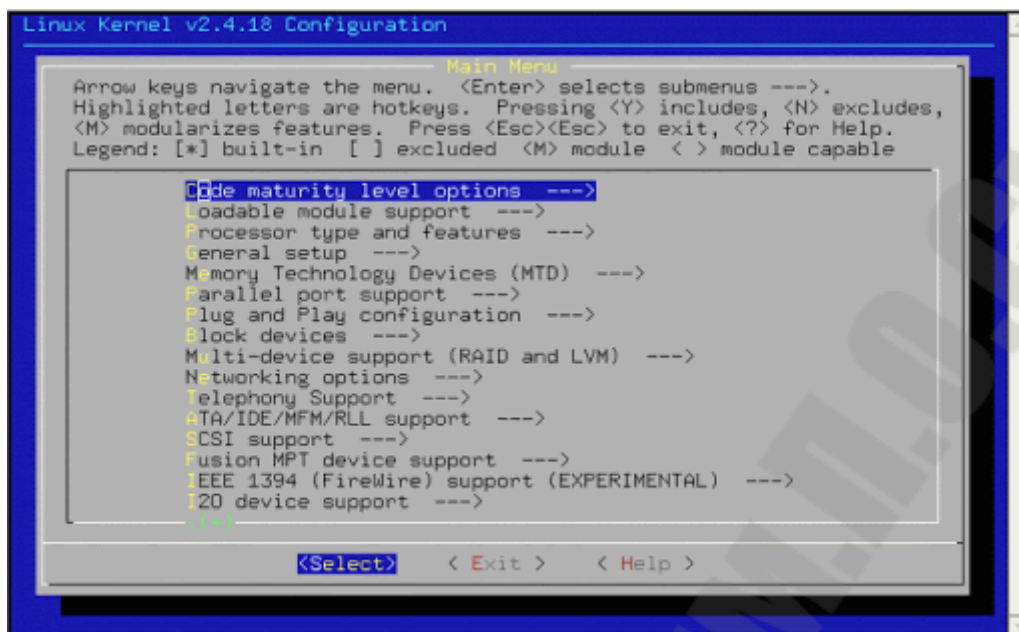


Рисунок 4.6. Внешний вид утилиты конфигурации ядра с псевдографическим интерфейсом

В ядре Linux предусмотрено два варианта сборки модулей, первый вариант импортирования модуля непосредственно в файл образа ядра, а второй сборка модуля в виде отдельного файла. Некоторые модули могут быть собраны как по первому, так и по второму варианту, а некоторые только в одном варианте. Если рассмотреть рис. 3 повнимательнее, то можно заметить, что возле некоторых пунктов в области CheckBox установлена галочка, а возле других установлена точка. Галочка означает что модуль будет собран и импортирован непосредственно в образ ядра, а точка обозначает что модуль будет собран в виде отдельного файла. Если в области CheckBox нет никакой отметки, то это означает что данный модуль не будет собран.

В данном примере необходимо включить сборку следующих модулей:

- USBGadgetfunctionsconfigurablethroughconfigs (USB\_CONFIGFS), данный модуль осуществляет поддержку USB устройств;
- HIDfunction (USB\_CONFIGFS\_F\_HID), данный модуль осуществляет поддержку HID устройств;
- Massstorage (USB\_CONFIGFS\_MASS\_STORAGE), данный модуль осуществляет поддержку USB накопителей, таких как флешкарты, внешние винчестеры и т.д.;
- RNDISsupport (USB\_ETH\_RNDIS), данный модуль отвечает за поддержку системы виртуальных сетевых карт, он необходим для возможности выхода в интернет через модуль N720;

- MultifunctionCompositeGadget (USB\_G\_MULTI), данный модуль обеспечивает поддержку виртуального подключения Ethernet;
- FunctionFilesystem (USB\_FUNCTIONFS), данный модуль обеспечивает поддержку файловых систем на USB устройства;
- USBFTDISinglePortSerialDriver (USB\_SERIAL\_FTDI\_SIO), данный модуль обеспечивает поддержку USB-UART преобразователей на чипе FT232;
- USBProlific 2303 SinglePortSerialDriver (USB\_SERIAL\_PL2303), данный модуль обеспечивает поддержку USB-UART преобразователей на чипе PL2303;
- USBWinchipheadCH341 SinglePortSerialDriver (USB\_SERIAL\_CH341), данный модуль обеспечивает поддержку USB-UART преобразователей на чипе CP341;
- USBCP210xfamilyofUARTBridgeControllers (USB\_SERIAL\_CP210X), данный модуль обеспечивает поддержку USB-UART преобразователей на чипе CP210X.

Может возникнуть вопрос: каким образом собирать тот или иной модуль, если он поддерживает оба варианта сборки? Ответ на данный вопрос вполне логичен и прост: Если модуль, обеспечивающий ту или иную функцию будет часто применяться в процессе работы устройства, то стоит его собрать в качестве встраиваемого элемента ядра (другими словами чтобы данный модуль сразу был импортирован в файл образа ядра), это повысит скорость работы с данным устройством, но при этом и увеличит объем памяти, занимаемый файлом образа ядра. Те модули, которые будут использоваться редко стоит собирать в виде отдельных файлов.

После выполнения включения в сборку всех необходимых файлов нужно сохранить конфигурацию. Все изменения будут записаны в файл .config. Если вам важно сохранить данную конфигурацию для последующих сборок, то данный файл стоит скопировать в отдельную папку и присвоить ему имя, которое будет отражать суть данной конфигурации. В противном случае при очистке кода ядра от файлов предыдущей сборки, данная конфигурация будет удалена.

После того как конфигурация сохранена можно приступить к процессу сборки. Сборка производится в несколько этапов. Для выполнения сборки необходимо выполнить следующую последовательность команд:

```
make -j7 ARCH=arm CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi- zImage dtbs modules
```

Данной командой запускается процесс сборки файла образа ядра в сжатом виде, сборка дерева устройств и модулей.

```
make ARCH=arm CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi- modules_prepare
```

Данной командой запускается сборка внешних модулей. Если выполнение данных команд закончилось удачно, то в папке с исходным кодом появится образ ядра и все необходимые модули. После этого можно приступить к процессу установки. Для этого необходимо ассоциировать файл образа системного диска с блочным устройством, а так же смонтировать оба раздела в соответствующие папки. Данный процесс был подробно описан в предыдущих лабораторных работах.

Процесс установки так же производится в несколько этапов. Пусть в данном случае раздел загрузчика будет смонтирован в каталог /mnt/boot, а раздел корневой файловой системы в каталог /mnt/rootfs. Первым этапом производится копирование файла образа ядра на раздел загрузчика. Для этого выполняется команда `cp` с указанием расположения файла образа ядра и пути, по которому он должен быть скопирован:

```
sudo cp arch/arm/boot/zImage /mnt/boot
```

Аналогичным образом стоит поступить и с файлом дерева устройств:

```
sudo cp arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dtb /mnt/boot
```

Для установки модулей вызывается утилита `make` с правами `root`, ей задается целевая архитектура, указывается полный путь к кросскомпилятору, путь для установки модулей, а так же опция `modules_install`:

```
sudo make ARCH=arm CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi- INSTALL_MOD_PATH=/mnt/rootfs modules_install
```

Установка заголовочных файлов происходит аналогичным образом, только указывается опция `headers_install`:

```
sudo make ARCH=arm CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi- headers_install INSTALL_HDR_PATH=/mnt/rootfs/usr
```

Ну и таким же способом устанавливается программное обеспечение для аппаратных устройств (если в текущей конфигурации оно присутствует):

```
sudo make ARCH=arm CROSS_COMPILE=/opt/x-tools/arm-opiz-linux-gnueabi/bin/arm-opiz-linux-gnueabi- firmware_install INSTALL_FW_PATH=/mnt/rootfs/lib/firmware
```

При успешном выполнении всех этих команд можно считать, что собранное ядро успешно установилось. После этого необходимо произвести размонтирование обоих разделов и снятие ассоциации образа диска с блочным устройством.

### **3.2 Задание для самостоятельного выполнения**

Необходимо произвести сборку ядра Linux заданной версии под заданную платформу с заданным перечнем периферийных устройств, а так же осуществить его установку в образ системного диска. Платформу, версию ядра и набор периферии задает преподаватель.

## **4. Содержание отчета**



Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а так же снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Так же обязательно выполнить примеры из данной работы и вставить снимки экрана процесса выполнения примеров. Вывод о полученных навыках и знаний после выполнения работы.

### **Контрольные вопросы для самопроверки**

1. Что такое ядро операционной системы?
2. Какими способами можно получить исходный код ядра Linux ?
3. В какой последовательности производится сборка ядра Linux?
4. Какие основные пути конфигурации сборки ядра Linux существуют?

## Лабораторная работа № 5

### Развертывание rootfs для целевой платформы

#### 1. Цель работы

Изучить основные принципы развертывание корневой файловой системы и окружения.

#### 2. Основные теоретические сведения

Существует множество способов развертывания корневой файловой системы и системного окружения, начиная от сборки каждого компонента вручную, заканчивая автоматизированными средствами типа BuildRoot. Метод развертывания выбирается в зависимости от задачи, однако большинство этих способов имеет один недостаток: отсутствие официальных репозиториях и невозможность обновления программных компонентов. В данной лабораторной работе будет рассмотрен наиболее простой способ — развертывание корневой файловой системы на основе ОС Ubuntu из готовых репозиториях. Данный способ лишен описанного недостатка, однако имеет другой: целевая архитектура должна поддерживаться официальными репозиториями Ubuntu.

##### 2.1 Основные сведения об ОС Ubuntu

Ubuntu — операционная система, основанная на Debian GNU/Linux. Основным разработчиком и спонсором является компания Canonical. Первоначальным именем проекта Ubuntu было No-Name-Yet («пока ещё нет имени»). Изначально Ubuntu создавалась как временное ответвление от Debian с целью регулярно выпускать новую версию операционной системы каждые шесть месяцев. В отличие от других ответвлений Debian общего назначения, таких как Xandros, Linspire и Libranet, Canonical осталась близка к философии Debian и включает в Ubuntu в основном свободное программное обеспечение вместо того, чтобы частично положиться на несвободные добавления. Пакеты Ubuntu по большей части базируются на пакетах из нестабильной (unstable) группы пакетов Debian. В Ubuntu используется Advanced Packaging Tool от Debian для управления установленными пакетами. Тем не менее, пакеты для Ubuntu и Debian не обязательно совместимы друг с другом. Некоторые разработчики Ubuntu также занимаются ключевыми пакетами Debian, поэтому в случае внесения изменений в собираемые программы они вносятся в оба проекта.

Ubuntu ориентирована на удобство и простоту использования. Она включает широко распространённое использование утилиты sudo, которая позволяет пользователям выполнять администраторские задачи, не запуская потенциально опасную сессию суперпользователя. Ubuntu, кроме того, имеет

развитую интернационализацию, обеспечивающую максимальную доступность для представителей разных языковых групп. С версии 5.04 кодировкой по умолчанию является UTF-8.

## 2.2 Утилита `debootstrap`

Утилита `debootstrap` - это инструмент, который может установить базовую систему Ubuntu в подкаталог другой, уже установленной системы. Он не требует инсталляционного CD и просто получает доступ к репозиторию Ubuntu. Он может также устанавливаться и управляться из другой операционной системы, таким образом, например, можно использовать `debootstrap`, чтобы установить Ubuntu на неиспользованный раздел из запущенной Gentoo системы. Он может также использоваться для создания `rootfs` для машин различной архитектуры. Это "cross-debootstrapping". Есть также в основном эквивалентная версия, написанная на C: `cdebootstrap`, который имеет меньший размер (табл. 5.1 и 5.2).

Установка из репозитория:  
`sudo apt-get install debootstrap`

Таблица 5.1. Основные параметры утилиты

Параметр	Описание
<code>--arch=ARCH</code>	Опция задает целевую архитектуру, если она отличается от текущей
<code>-include=alpha,beta</code>	Список дополнительных пакетов, которые необходимо установить. Элементы списка разделяются запятыми
<code>--exclude=alpha,beta</code>	Список пакетов, которые необходимо исключить.
<code>--variant=minbase ...</code>	Имя используемого варианта сценария загрузки. В настоящее время поддерживаются варианты: <ul style="list-style-type: none"> <li>☞ <code>minbase</code>, которые включают только необходимые пакеты и <code>apt</code>;</li> <li>☞ <code>buildd</code>, который устанавливает сборку необходимых пакетов в <code>TARGET</code>;</li> <li>☞ <code>fakechroot</code>, который устанавливает пакеты без привилегий <code>root</code>.</li> <li>☞ <code>scratchbox</code>, который предназначен для создания целей для использования с нуля.</li> </ul> По умолчанию, без аргумента <code>--variant = X</code> , устанавливается базовая версия <code>Dbeian</code> .
<code>--no-check-gpg</code>	Отключает проверку подписи <code>gpg</code> .
<code>--verbose</code>	Получите больше информации о загрузке.
<code>--download-only</code>	Загрузить пакеты без установки.

--no-check-certificate	Не проверять сертификаты.
--foreign	Выполняйте только начальную фазу начальной загрузки, например, если целевая архитектура не соответствует хосту архитектура.

Пример использования:

```
sudo debootstrap --arch=i386 --include=gcc,g++,git,nano artful /opt/chroot
ftp://ftp.byfly.by/ubuntu
```

Таблица 5.2. Версии Ubuntu и их кодовые имена

Версия	Кодовое имя	Окончание срока поддержки
12.04 LTS	Precise Pangolin	апрель 2017 года
12.10	Quantal Quetzal	16 мая 2014 года
13.04	Raring Ringtail	27 января 2014 года
13.10	Saucy Salamander	17 июля 2014 года
14.04 LTS	Trusty Tahr	апрель 2019 года
14.10	Utopic Unicorn	23 июля 2015 года
15.04	Vivid Vervet	4 февраля 2016 года
15.10	Wily Werewolf	июль 2016 года
16.04 LTS	Xenial Xerus	апрель 2021 года
16.10	Yakkety Yak	июль 2017 года
17.04	Zesty Zapus	январь 2018 года
17.10	Artful Aardvark	июль 2018 года
18.04	Bionic Beaver	Апрель 2023
18.10	Cosmic Cuttlefish	Июль 2019
19.04	Disco Dingo	23 января 2020 года
19.10	Eoan Ermine	июль 2020 года
20.04 LTS	Focal Fossa	апрель 2025 года
20.10	Groovy Gorilla	июль 2021 года

После окончания срока поддержки версии Ubuntu удаляются из официального репозитория и основных зеркал и перемещаются в <http://archive.ubuntu.com/ubuntu/> (табл. 5.3).

Поддерживаемые архитектуры (официальный репозиторий):

- ⑩ i386 (x86 intel)
- ⑩ x86\_64 (amd64)
- ⑩ arm
- ⑩ armhf (hard float)
- ⑩ ppc64el(OpenPOWER)
- ⑩ S390X (IBM Mainframe Systems)

Таблица 5.3. Зеркала репозитория Ubuntu

Ссылка	Расположение
<a href="http://ftp.byfly.by/ubuntu/">http://ftp.byfly.by/ubuntu/</a>	Беларусь
<a href="http://mirror.datacenter.by/ubuntu/">http://mirror.datacenter.by/ubuntu/</a>	
<a href="http://mirror.yandex.ru/ubuntu/">http://mirror.yandex.ru/ubuntu/</a>	Россия
<a href="http://mirror.truenetwork.ru/ubuntu/">http://mirror.truenetwork.ru/ubuntu/</a>	
<a href="http://mirror.corbina.net/ubuntu/">http://mirror.corbina.net/ubuntu/</a>	
<a href="http://ubuntu.volia.net/ubuntu-archive/">http://ubuntu.volia.net/ubuntu-archive/</a>	Украина
<a href="http://ubuntu.ip-connect.vn.ua/">http://ubuntu.ip-connect.vn.ua/</a>	
<a href="http://ubuntu.mirrors.omnilance.com/ubuntu/">http://ubuntu.mirrors.omnilance.com/ubuntu/</a>	
<a href="http://mirror.vpsnet.com/ubuntu/">http://mirror.vpsnet.com/ubuntu/</a>	Литва
<a href="http://ubuntu-archive.mirror.serveriai.lt/">http://ubuntu-archive.mirror.serveriai.lt/</a>	
<a href="http://ubuntu.mirror.vu.lt/ubuntu/">http://ubuntu.mirror.vu.lt/ubuntu/</a>	
<a href="http://ftp.icm.edu.pl/pub/Linux/ubuntu/">http://ftp.icm.edu.pl/pub/Linux/ubuntu/</a>	Польша
<a href="http://mirror.onet.pl/pub/mirrors/ubuntu/">http://mirror.onet.pl/pub/mirrors/ubuntu/</a>	
<a href="http://ftp.agh.edu.pl/ubuntu/">http://ftp.agh.edu.pl/ubuntu/</a>	

Но не все репозитории поддерживают полный список архитектур. Информация по каждому репозиторию можно посмотреть на сайте <https://launchpad.net/ubuntu/+archivemirrors>.

Если не указывается явным образом ссылка на репозиторий, то скачивание происходит из основного репозитория Ubuntu.

### 2.3 Утилита chroot

Chroot — операция изменения корневого каталога диска для запущенного процесса и его дочерних процессов. Программа, запущенная в таком окружении не может получить доступ к файлам вне нового корневого каталога. Это измененное окружение называется chroot jail.

Синтаксис использования:

```
chroot новый_корень [команда [аргументы]...]
```

```
chroot опции
```

Утилита `chroot` запускает на выполнение команду с указанным каталогом `новый_корень` в качестве корневого. На многих системах, это доступно только суперпользователю.

Обычно, имена файлов начинаются от корня дерева каталогов, т.е. `"/`. Утилита `chroot` изменяет корень на каталог `новый_корень` (который должен существовать) и затем выполняет команду с её [аргументами] (наличие последних необязательно). Если команда не задана, то будет вызвана командная оболочка `"${SHELL} -i"` (по умолчанию `"/bin/sh"`, если переменная среды окружения не установлена).

## 2.4 Пакет `qemu-user-static`

Пакет `qemu-user-static` содержит в себе набор утилит для трансляции команд целевой архитектуры. Грубо говоря данный пакет представляет собой самый простой эмулятор какой либо архитектуры и позволяет запускать программы собранные например под архитектуру ARM на архитектуре x86. Конечно, как и в случае полноценной виртуальной машины происходит потеря производительности, по сравнению с работой аналогичных программ но на родной архитектуре, однако данный пакет очень сильно выручает при развертывании корня файловой системы и окружения под несовместимую архитектуру. Данный пакет содержит следующий перечень программ:

- `qemu-aarch64_be-static`
- `qemu-i386-static`
- `qemu-mipsn32-static`
- `qemu-riscv32-static`
- `qemu-tilegx-static`
- `qemu-aarch64-static`
- `qemu-m68k-static`
- `qemu-mips-static`
- `qemu-riscv64-static`
- `qemu-x86_64-static`
- `qemu-alpha-static`
- `qemu-microblazeel-static`
- `qemu-nios2-static`
- `qemu-s390x-static`
- `qemu-xtensaeb-static`

- qemu-armeb-static
- qemu-microblaze-static
- qemu-or1k-static
- qemu-sh4eb-static
- qemu-xtensa-static
- qemu-arm-static
- qemu-mips64el-static
- qemu-ppc64abi32-static
- qemu-sh4-static
- qemu-cris-static
- qemu-mips64-static
- qemu-ppc64le-static
- qemu-sparc32plus-static
- qemu-debootstrap
- qemu-mipsel-static
- qemu-ppc64-static
- qemu-sparc64-static
- qemu-hppa-static
- qemu-mipsn32el-static
- qemu-ppc-static
- qemu-sparc-static

Как видно из данного списка, перечень поддерживаемых архитектур довольно обширен.

Таблица 5.4. Основные опции программ пакета qemu-user-static

Опция	Описание
-h	Вывод справки
-g	Ожидание подключения отладчика к порту 1234
-L <path>	Задать путь эмулятора для исполняемого файла (default=/usr/gnemul/qemu-arm)
-s <size>	Установить размер стека в байтах (default=524288)
-d <options>	Указать путь к логу
-p <pagesize>	Установить размер страницы

### 3. Порядок выполнения работы

#### 3.1 Пример развертывания и настройки корневой файловой системы и окружения для Orange Pi Zero

Все действия по установке и развертыванию системы должны производиться на разделах образа системного диска, создание которого было описано ранее, поэтому перед проведением все манипуляций по развертыванию необходимо ассоциировать образ системного диска с блочным устройством и произвести монтирование разделов этого образа.

В первую очередь необходимо установить необходимые пакеты. Утилита `chroot` обычно по умолчанию входит в состав `Ubuntu`, а пакеты `qemu-user-static` и `debootstrap` необходимо ставить отдельно. Установить данные пакеты можно с помощью следующей команды:

```
sudo apt install debootstrap qemu-user-static
```

Далее процесс будет очень похож на развертывание изолированной среды, однако целевая платформа будет иметь архитектуру `armhf` и потребуются произвести дополнительные настройки, чтобы данная среда смогла работать самостоятельно как полноценная операционная система. В данном примере, для упрощения работы будут применяться переменные окружения. В первую очередь создается переменная окружения, которая будет содержать кодовое имя дистрибутива:

```
export Distro="xenial"
```

Как видно из кодового имени, в данном примере будет развертываться `Ubuntu 16.04`. Затем можно запустить утилиту `debootstrap` с указанием ряда параметров:

```
sudo debootstrap --arch=armhf --include=gcc,g++,git,nano,make,screen,ssh,net-tools --foreign $Distro /mnt/rootfs
```

В данном случае набор параметров задает целевую архитектуру `armhf`, версия дистрибутива задается с помощью переменной окружения, а так же перечень устанавливаемых дополнительных пакетов. Отдельно стоит обратить внимание на параметр `--foreign`. Как видно из таблицы 1 данный параметр указывает на то, что будет произведен только первый шаг развертывания. Это необходимо т. к. Целевая архитектура является несовместимой, и вторая база развертывания будет производиться с использованием эмулятора. При удачном выполнении данной команды в указанную папку будет произведено скачивание всех необходимых архивов.

Для продолжения развертывания необходимо скопировать эмулятор в каталог, в котором производится развертывание:

```
sudo cp /usr/bin/qemu-arm-static /mnt/rootfs/usr/bin/
```

После этого можно выполнить `chroot` в указанную папку. Однако стоит учесть два момента. Первый заключается в том, что архитектура несовместимая а второй в том что целевая система еще не развернута. Команда будет выглядеть следующим образом:

```
sudo chroot /mnt/rootfs /usr/bin/qemu-arm-static /bin/sh -i
```

В данной команде утилите `chroot` задается каталог, в котором развертывается операционная система, так же указывается путь к эмулятору и командная оболочка которая будет запущена. После выполнения смены корневой файловой системы и запуска командной оболочки необходимо



выполнить второй шаг развертывания. Сделать это можно следующей командой:

```
/debootstrap/debootstrap --second-stage
```

Параметр `--second-stage` указывает на то, что необходимо выполнить второй шаг развертывания. После удачного выполнения данной команды в указанной папке будет развернута полноценная операционная система по заданную архитектуру, однако для того чтобы она могла работать самостоятельно, необходимо произвести некоторые настройки. Повторно задаем переменную окружения которая задает кодовое имя дистрибутива (это необходимо потому, что предыдущая переменная окружения в среде `chroot` уже не действует):

```
export Distro="xenial"
```

Для возможности установки пакетов, необходимо создать список со ссылками на них:

```
cat <<EOT > /etc/apt/sources.list
deb http://ports.ubuntu.com/ $Distro main restricted
deb http://ports.ubuntu.com/ $Distro-updates main restricted
deb http://ports.ubuntu.com/ $Distro universe
deb http://ports.ubuntu.com/ $Distro-updates universe
deb http://ports.ubuntu.com/ $Distro multiverse
deb http://ports.ubuntu.com/ $Distro-updates multiverse
deb http://ports.ubuntu.com/ $Distro-backports main restricted universe multiverse
#deb http://ports.ubuntu.com/ $Distro partner
deb http://ports.ubuntu.com/ $Distro-security main restricted
deb http://ports.ubuntu.com/ $Distro-security universe
deb http://ports.ubuntu.com/ $Distro-security multiverse
EOT
```

Так же необходимо прописать правила монтирования разделов:

```
cat <<EOT >> /etc/fstab
/dev/mmcblk0p2 / ext4 rw,relatime,data=ordered 0 1
/dev/mmcblk0p1 /boot vfat
rw,relatime,fmask=0022,dmask=0022,codepage=437,ioccharset=iso8859-
1,shortname=mixed,errors=remount-ro 0 2
EOT
```

В данном случае прописывается монтирование двух разделов. Первый раздел `/dev/mmcblk0p1`, содержащий настройки `u-boot` и ядро операционной системы, будет монтироваться в каталог `/boot`. А второй раздел `/dev/mmcblk0p2` будет являться корнем файловой системы.

После проведения первичных настроек необходимо выйти из данной сессии `chroot` с помощью команды `exit`. Далее можно приступить к установке дополнительных программных пакетов. Однако для этого системе потребуется доступ в интернет. Для предоставления возможности выхода в интернет гостевой операционной системе необходимо скопировать файл настроек DNS серверов `/etc/resolv.conf` в соответствующий каталог гостевой операционной системы, а так же смонтировать специализированные

файловые системы. Реализовать это можно с помощью следующей последовательности команд:

```
sudo cp /etc/resolv.conf /mnt/rootfs/etc
sudo mount --bind /proc /mnt/rootfs/proc
sudo mount --bind /sys /mnt/rootfs/sys
sudo mount --bind /dev /mnt/rootfs/dev
```

После этого можно открывать новую сессию chroot и приступать к настройкам системы и установке дополнительных программных пакетов. В данном случае уже нет необходимости в явном виде указывать командную оболочку и эмулятор:

```
sudo chroot /mnt/rootfs
```

После этого должна автоматически запуститься командная оболочка bash. Первое, что стоит сделать, это задать системный язык и кодировку. Для удобства задается кодировка UTF8 и русский язык. Для этого устанавливаются значения соответствующих системных переменных:

```
export LANG=ru_RU.UTF-8
export LC_ALL=ru_RU.UTF-8
```

После этого в обязательном порядке необходимо выполнить обновление программных пакетов:

```
apt-get update
apt-get upgrade
```

Обычно при первом обновлении количество скачиваемых пакетов велико и данный процесс может занять определенное время. После установки обновлений пакетов необходимо установить пакет локализаций:

```
apt-get install locales
dpkg-reconfigure locales
sudo apt-get install language-pack-ru
sudo update-locale LANG=ru_RU.UTF-8
```

После установки пакета локализаций интерфейс должен перейти полностью на русский язык. После этого можно установить минимальный набор необходимых пакетов. К таким пакетам можно отнести компиляторы gcc и g++, монитор процессов htop, консольный текстовый редактор nano и openssh-server. Для установки перечисленных пакетов можно воспользоваться следующей командой:

```
apt install gcc g++ mchtop nano openssh-server
```

Обязательным требованием безопасности является установка хорошего пароля для пользователя root. Установить параметр можно следующей командой:

```
passwd root
```

Стоит отметить, что при вводе пароля символы не будут отображаться. Затем, опять же по требованиям безопасности, следует создать обычного пользователя:

```
adduser User
```

При выполнении данной команды будет произведен запрос пароля для данного пользователя, а так же различная информация. Обязательным для

ввода является только пароль. Чтобы данный пользователь имел возможность использовать утилиту sudo необходимо добавить соответствующую запись в файл /etc/sudoers. Необходимо открыть для редактирования данный файл:

```
nano /etc/sudoers
```

А затем в него внести запись UserALL=(ALL:ALL) ALL.

Для поддержки работы сети необходимо присвоить устройству сетевое имя, а так же выполнить настройку сетевого интерфейса. Для задания сетевого имени устройства необходимо открыть файл /etc/hostname и первой же строкой записать в него имя устройства:

```
sudo nano /etc/hostname
```

И записать в данный файл имя устройства user-rc. Для настройки сетевого интерфейса необходимо отредактировать файл /etc/network/interfaces. В данном примере будет выполнена настройка интерфейса Ethernet на автоматическое подключение и получение ip адреса с сетевого сервера DHCP:

```
cat <<EOT >> /etc/network/interfaces
```

```
auto lo eth0
```

```
iface lo inet loopback
```

```
iface eth0 inet dhcp
```

```
EOT
```

Помимо этого, для возможности работы с доменными именами, а так же для доступа в интернет стоит прописать настройки серверов доменных имен:

```
cat <<EOT >> /etc/resolv.conf
```

```
nameserver 8.8.8.8
```

```
nameserver 4.4.4.4
```

```
nameserver 192.168.1.254
```

```
EOT
```

В данном случае прописывается два сервера доменных имен от компании Google и один локальный сервер. На этом основную настройку можно считать законченной. Если требуется настройка каких либо дополнительных параметров, то можно их произвести. После окончания настройки необходимо выйти из сессии chroot, а так же в обязательном порядке размонтировать специальные файловые системы:

```
exit
```

```
sudo umount /mnt/rootfs/proc
```

```
sudo umount /mnt/rootfs/sys
```

```
sudo umount /mnt/rootfs/dev
```

После этого можно размонтировать разделы образа системного диска и снять ассоциацию блочного устройства.

#### **4. Содержание отчета**

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а так же

снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Так же обязательно выполнить примеры из данного работы и вставить снимки экрана процесса выполнения примеров. Вывод о полученных навыках и знаний после выполнения работы.

### **Контрольные вопросы для самопроверки**

1. Что такое Ubuntu?
2. Какие версии Ubuntu существуют?
3. Какие две основных группы версий Ubuntu существуют и в чем между ними различия?
4. На какие этапы делится процесс развертывания корневой файловой системы и системного окружения?

## Литература

1. Блум, Р. Командная строка Linux и сценарии оболочки : библия пользователя / Ричард Блум, Кристина Бреснахэн ; пер. с англ. и ред. К. А. Птицина. - 2-е изд.. - Москва [и др.] : Диалектика, 2013. - 784 с. УДК 004.451 ББК 32
2. Донцов, В. П. LINUX на примерах / В. П. Донцов, И. В. Сафин. - Санкт-Петербург : Наука и техника, 2017. - 346 с.. - (Просто о сложном) УДК 004.451 ББК 32
3. Иванов Н. Н. Программирование в Linux. - Санкт-Петербург : БХВ-Петербург, 2007. - 402с. + Компакт-диск. - (Самоучитель) УДК 004.451 ББК 32
4. Кетов, Д. В. Внутреннее устройство Linux / Д. В. Кетов. - Санкт-Петербург : БХВ-Петербург, 2017. - 307 с. УДК 004.451.9Linux ББК 32
5. Колисниченко, Д. Н. Linux-сервер своими руками : полное руководство / Д. Н. Колисниченко. - Санкт-Петербург : Наука и техника, 2008. - 618 с. + CD УДК 004.451 ББК 32
6. Колисниченко Д. Н. Самоучитель Linux. - 2-е изд.. - Санкт-Петербург : БХВ-Петербург, 2008. - 429 с. УДК 004.451
7. Колисниченко Д. Н. Серверное применение Linux. - Санкт-Петербург : БХВ-Петербург, 2008. - 509с. УДК 004.451
8. Лав Р. LINUX. Системное программирование. - Санкт-Петербург : Питер, 2008. - 413 с. УДК 004.451 ББК 32
9. Маслаков, В. Г. Linux / В. Маслаков. - Санкт-Петербург : Питер, 2009. - 330 с. + DVD. - (На 100%) УДК 004.451 ББК 32
10. Операционная система LINUX для начинающих и не только : Кратко, доступно, просто / Сергей Ивановский. - Москва : Познавательная книга плюс, 1999. - 192 с УДК 004.451 ББК 32
11. Рейчард К. LINUX: справочник. - 2-е изд.. - СПб : Питер Ком, 1999. - 480с. УДК 004.451(035) ББК 32

## Содержание

	Стр.
Лабораторная работа № 1. Применение cetti для кросса компиляции Консольный тонкий клиент.....	3
Лабораторная работа № 2. Сборка кросс компилятора с помощью crosstool-ng .....	22
Лабораторная работа № 3. Сборка загрузчика U-Boot под заданную платформу.....	55
Лабораторная работа № 4. Сборка ядра и модулей по заданную платформу .....	68
Лабораторная работа № 5. Развертывание rootfs для целевой платформы.....	79
Литература.....	90

**Сахарук Андрей Владимирович**

**ПРОЕКТИРОВАНИЕ УПРАВЛЯЮЩИХ  
И ИНФОРМАЦИОННЫХ СРЕДСТВ  
НА БАЗЕ ЕМБЕДДЕД СИСТЕМ**

**Практикум  
по выполнению лабораторных работ  
по одноименной дисциплине для студентов  
специальности 1-36 01 01 «Информационные  
технологии и управление в технических системах»  
дневной формы обучения**

Подписано к размещению в электронную библиотеку  
ГГТУ им. П. О. Сухого в качестве электронного  
учебно-методического документа 31.03.21.

Пер. № 66Е.

<http://www.gstu.by>