

Содержание

Предисловие	15
Об авторе	16
О рецензентах.....	17
Вступление	19
Глава 1. Приступая к работе	25
Выбор правильной операционной системы.....	26
Игроки	27
Жизненный цикл проекта.....	28
Четыре составные части встраиваемой Linux-системы	29
Программное обеспечение с открытым исходным кодом	30
Лицензии	30
Оборудование для встраиваемых Linux-систем	31
Используемое оборудование.....	33
Плата BeagleBone Black	33
QEMU	34
Используемое программное обеспечение	35
Резюме	35
Глава 2. О наборах инструментов	36
Что такое набор инструментов?.....	36
Типы наборов инструментов – платформенные и перекрестные	38
Архитектура процессора	39
Выбор библиотеки C.....	40
Получение набора инструментов	42
Сборка набора инструментов с помощью crosstool-NG	43
Установка crosstool-NG.....	43
Выбор набора инструментов.....	44
Анатомия набора инструментов	46
Получение информации о кросс-компиляторе.....	46
sysroot, библиотека и файлы-заголовки	48
Другие элементы набора инструментов.....	48
Компоненты библиотеки C	49
Статическая и динамическая компоновка с библиотеками.....	50
Статические библиотеки.....	50
Разделяемые библиотеки.....	51

Искусство кросс-компиляции.....	53
Простые make-файлы.....	54
Autotools.....	54
Конфигурирование пакета.....	57
Проблемы кросс-компиляции.....	58
Резюме.....	58
Глава 3. Все о начальных загрузчиках	60
Что делает начальный загрузчик?.....	60
Последовательность начальной загрузки.....	61
Этап 1: код в ПЗУ.....	62
Этап 2: SPL.....	63
Этап 3: TPL.....	63
Загрузка из UEFI-прошивки.....	64
Переход от начального загрузчика к ядру.....	65
Введение в деревья устройств.....	66
Основные сведения о деревьях устройств.....	66
Свойство reg.....	67
Указатели на описатели и прерывания.....	68
Включаемые файлы деревьев устройств.....	69
Компиляция дерева устройств.....	71
Выбор начального загрузчика.....	71
U-Boot.....	72
Сборка U-Boot.....	72
Установка U-Boot.....	73
Работа с U-Boot.....	74
Загрузка Linux.....	78
Kconfig и U-Boot.....	79
Сборка и тестирование.....	82
Режим Сапсан.....	82
Varebox.....	82
Получение Varebox.....	83
Сборка Varebox.....	83
Резюме.....	84
Глава 4. Портирование и конфигурирование ядра	86
Что делает ядро?.....	86
Выбор ядра.....	88
Цикл разработки ядра.....	88
Стабильные и долгосрочные версии.....	89
Поддержка со стороны производителя.....	90
Сборка ядра.....	90
Получение исходного кода.....	90

О конфигурировании ядра	91
Использование переменной LOCALVERSION для идентификации ядра.....	95
Модули ядра	96
Компиляция.....	96
Компиляция образа ядра.....	97
Компиляция деревьев устройств	99
Компиляция модулей.....	99
Удаление артефактов сборки	99
Загрузка ядра.....	100
BeagleBone Black.....	100
QEMU.....	100
Паника ядра	101
Подготовка пользовательского пространства	102
Сообщения ядра	102
Командная строка ядра.....	103
Портирование Linux на новую плату.....	104
С деревом устройств	104
Без дерева устройств	105
Дополнительная литература.....	107
Резюме	107

Глава 5. Построение корневой файловой системы 109

Что должно быть в корневой файловой системе?	109
Структура каталогов.....	111
Каталог технологической подготовки	111
Программы в корневой файловой системе	114
Программа init	114
Оболочка.....	115
Утилиты.....	115
BusyBox спешит на помощь!.....	115
ToyBox – альтернатива BusyBox	117
Библиотеки для корневой файловой системы	118
Уменьшение размера путем удаления таблицы символов	119
Узлы устройств.....	119
Файловые системы proc и sysfs	120
Монтирование файловых систем.....	121
Модули ядра	122
Перенос корневой файловой системы на целевое устройство.....	122
Создание загрузочного ram-диска.....	123
Автономный ram-диск.....	123
Загрузка ram-диска	124
Встраивание ram-диска в формате срjо в образ ядра	125
Старый формат initrd	126

Программа init.....	126
Конфигурирование учетных записей пользователей.....	127
Добавление учетных записей пользователей в корневую файловую систему.....	129
Запуск процесса-демона.....	129
Улучшенный способ управления узлами устройств.....	129
Пример использования devtmpfs.....	130
Пример использования mdev.....	130
А так ли плохи статические узлы устройств?.....	131
Конфигурирование сети.....	131
Сетевые компоненты для glibc.....	132
Создание образов файловой системы с помощью таблиц устройств.....	133
Копирование корневой файловой системы на карту SD.....	134
Монтирование корневой файловой системы по NFS.....	134
Тестирование в эмуляторе QEMU.....	135
Тестирование с платой BeagleBone Black.....	136
Проблемы с правами доступа к файлам.....	136
Загрузка ядра по протоколу TFTP.....	137
Дополнительная литература.....	138
Резюме.....	138

Глава 6. Выбор системы сборки 139

Довольно самопальных встраиваемых систем.....	139
Системы сборки.....	139
Форматы пакетов и менеджеры пакетов.....	141
Buildroot.....	141
История.....	142
Стабильные версии и поддержка.....	142
Установка.....	142
Конфигурирование.....	143
Выполнение.....	144
Создание специального BSP-пакета.....	145
Добавление своего кода.....	146
Соответствие лицензионным требованиям.....	148
Yocto Project.....	149
История.....	149
Стабильные версии и поддержка.....	150
Установка Yocto Project.....	151
Конфигурирование.....	151
Сборка.....	152
Выполнение.....	153
Слои.....	154
Настройка образов с помощью local.conf.....	159
Рецепт создания образа.....	159

Создание SDK.....	160
Контроль лицензий.....	162
Дополнительная литература.....	162
Резюме.....	162
Глава 7. Выбор стратегии хранения	164
Типы запоминающих устройств.....	164
Флэш-память типа NOR.....	165
NAND-память.....	166
Управляемая флэш-память.....	168
Доступ к флэш-памяти из начального загрузчика.....	169
U-Boot и флэш-память типа NOR.....	170
U-Boot и флэш-память типа NAND.....	170
U-Boot и карты MMC, SD и eMMC.....	170
Доступ к флэш-памяти из Linux.....	170
Устройства на основе технологии памяти.....	171
Драйвер блочного устройства MMC.....	176
Файловые системы для флэш-памяти.....	177
Уровень флэш-преобразования.....	177
Файловые системы для флэш-памяти типа NOR и NAND.....	178
JFFS2.....	178
YAFFS2.....	181
UBI и UBIFS.....	182
Файловые системы для управляемой флэш-памяти.....	186
Flashbench.....	187
Discard и TRIM.....	188
Ext4.....	189
F2FS.....	190
FAT16/32.....	190
Сжатые неизменяемые файловые системы.....	191
squashfs.....	191
Временные файловые системы.....	192
Превращение корневой файловой системы в неизменяемую.....	193
Варианты выбора файловой системы.....	194
Обновление в месте эксплуатации.....	194
Степень детализации: файл, пакет или образ?.....	195
Атомарное обновление образа.....	196
Дополнительная литература.....	197
Резюме.....	198
Глава 8. Введение в драйверы устройств	199
Роль драйверов устройств.....	199
Символьные устройства.....	200

Блочные устройства.....	202
Сетевые устройства.....	203
Получение информации о драйверах на этапе выполнения.....	205
Получение информации из sysfs.....	207
Устройства: /sys/devices.....	207
Драйверы: /sys/class.....	208
Блочные устройства: /sys/block.....	209
Поиск подходящего драйвера устройства.....	209
Драйверы устройств в пользовательском пространстве.....	210
GPIO.....	210
Светодиоды.....	213
Шина I2C.....	214
Шина SPI.....	216
Написание драйвера устройства.....	216
Проектирование интерфейса символьного устройства.....	216
Анатомия драйвера устройства.....	218
Загрузка модулей ядра.....	222
Определение конфигурации оборудования.....	222
Деревья устройств.....	223
Платформенные данные.....	223
Связывание оборудования с драйверами.....	224
Дополнительная литература.....	226
Резюме.....	226

Глава 9. Инициализация системы – программа init 228

После того как ядро загрузилось.....	228
Введение в программы init.....	229
BusyBox init.....	229
Скрипты инициализации в Buildroot.....	231
System V init.....	231
inittab.....	233
Скрипты init.d.....	235
Добавление нового демона.....	235
Запуск и остановка служб.....	236
systemd.....	237
Сборка systemd в Yocto Project и Buildroot.....	237
Как systemd загружает систему.....	239
Добавление своей службы.....	240
Добавление сторожевого таймера.....	241
Применение во встраиваемых Linux-системах.....	242
Дополнительная литература.....	243
Резюме.....	243

Глава 10. Процессы и потоки	244
Процесс или поток?.....	244
Процессы.....	246
Создание нового процесса	246
Завершение процесса	247
Выполнение другой программы	249
Демоны	251
Межпроцессное взаимодействие.....	251
Потоки	256
Создание нового потока	256
Завершение потока.....	258
Компиляция многопоточной программы.....	258
Межпроцессное взаимодействие.....	258
Мьютексы	259
Изменение условий.....	259
Разбиение проблемы на части.....	260
Планирование	262
Справедливость и детерминированность.....	262
Политики с разделением времени	263
Политики реального времени.....	264
Выбор политики.....	265
Выбор приоритета реального времени.....	266
Дополнительная литература.....	266
Резюме	267
 Глава 11. Управление памятью	 268
Основы виртуальной памяти.....	268
Структура памяти ядра.....	269
Сколько памяти потребляет ядро?	270
Структура памяти в пользовательском пространстве	272
Карта памяти процесса	274
Подкачка	275
Выгрузка страниц в сжатую память (zram).....	275
Отображение памяти с помощью mmap.....	276
Использование mmap для выделения частной памяти	276
Использование mmap для разделения памяти	276
Использование mmap для доступа к памяти устройства.....	277
Сколько памяти потребляет мое приложение?.....	277
Потребление памяти на уровне процесса.....	278
Использование top и ps.....	278
Использование smem.....	279
Другие инструменты.....	281

Обнаружение утечек памяти.....	281
mtrace	281
Valgrind.....	282
Нехватка памяти	283
Дополнительная литература.....	285
Резюме	285
Глава 12. Отладка в GDB	287
Отладчик GNU.....	287
Подготовка к отладке.....	287
Отладка приложений в GDB	288
Удаленная отладка с помощью gdbserver	289
Настройка Yocto Project	290
Настройка Buildroot	290
Начало отладки.....	290
Подключение GDB к gdbserver.....	290
Задание sysroot	291
Командные файлы GDB.....	292
Обзор команд GDB	292
Выполнение до точки прерывания	293
Отладка разделяемых библиотек	294
Yocto Project.....	294
Buildroot.....	295
Другие библиотеки.....	295
Своевременная отладка	295
Отладка разветвлений и потоков	296
Core-файлы	296
Использование GDB для анализа core-файлов	298
Пользовательские интерфейсы к GDB	298
Терминальный пользовательский интерфейс	299
Отладчик DDD.....	299
Eclipse.....	300
Отладка кода ядра.....	301
Отладка кода ядра в kgdb.....	301
Пример сеанса отладки.....	302
Отладка на ранних стадиях.....	304
Отладка модулей.....	304
Отладка кода ядра в kdb.....	305
Сообщения об ошибках ядра	306
Сохранение сообщения об ошибке ядра.....	307
Дополнительная литература.....	308
Резюме	309

Глава 13. Профилирование и трассировка	310
Эффект наблюдателя.....	310
Таблицы символов и флаги компиляции	311
Приступая к профилированию	311
Профилирование с помощью <code>top</code>	312
Профилировщик для бедных	313
Применение <code>perf</code>	314
Конфигурирование ядра для работы с <code>perf</code>	314
Сборка <code>perf</code> в Yocto Project.....	315
Сборка <code>perf</code> в Buildroot.....	315
Профилирование с помощью <code>perf</code>	315
Графы вызовов	317
<code>perf annotate</code>	318
Другие профилировщики: OProfile и gprof	319
Трассировка событий.....	321
Введение в Ftrace	321
Подготовка к работе с Ftrace	322
Динамический режим Ftrace и фильтры трассировки	324
События трассировки.....	325
Использование LTTng.....	326
LTTng и Yocto Project	327
LTTng и Buildroot	327
Применение LTTng для трассировки ядра	327
Использование Valgrind для профилирования приложений.....	330
Callgrind.....	330
Helgrind	330
Использование <code>strace</code> для показа системных вызовов.....	331
Резюме	333
Глава 14. Программирование в режиме реального времени.....	335
Что такое реальное время?	335
Определение источников недетерминированности	337
Задержки планирования	339
Вытеснение в ядре	340
Ядро Linux реального времени (PREEMPT_RT).....	340
Потоковые обработчики прерываний.....	341
Вытесняемые блокировки ядра.....	343
Получение заплат PREEMPT_RT	344
Yocto Project и PREEMPT_RT	344
Таймеры высокого разрешения.....	345
Предотвращение страничных отказов в приложении реального времени.....	345

Экранирование прерываний	346
Измерение задержек планирования	347
cyclictest	347
Ftrace	350
Комбинирование cyclictest и Ftrace	352
Дополнительная литература	352
Резюме	353
Предметный указатель	354

Предисловие

Linux – исключительно гибкая и мощная операционная система, и мне кажется, что мы еще не в полной мере осознали, какие преимущества сулит ее встраивание. Одна из возможных причин – многогранность этой темы и сложность ее изучения.

Конечно, можно самостоятельно пролагать себе путь в мире встраиваемых Linux-систем, как я и поступал в течение десяти лет, но отрадно видеть, что такие люди, как Крис, пишут книги, помогающие другим познакомиться со многими полезными вещами. Уж я-то точно нашел бы подобной книжке применение, если бы она существовала, когда я только начинал!

Понятно, что у меня есть личный интерес к проекту Yocto Project, поскольку это мое основное занятие и попытка внести свой вклад в мир встраиваемых Linux-систем. Одна из его основных целей – облегчить жизнь тем, кто создает такие системы. И мы добились кое-каких успехов, но есть области, требующие дальнейшей работы. Мы неустанно стремимся снять барьеры на пути начинающих и расширить круг посвященных, сделать технологию более доступной и распространенной.

В своей книге Крис преследует те же цели. Надеюсь, что вам понравится и книга, и сама система Linux и что в конечном итоге вы вольетесь в одно из активных сообществ, сформировавшихся вокруг проектов с открытым исходным кодом, стоящих за многими компонентами, о которых пойдет речь.

Ричард Пэрдай,
архитектор Yocto Project, член фонда Linux Foundation

Об авторе

Крис Симмондс – консультант по программному обеспечению и преподаватель, проживает в южной части Англии. С конца 1990-х годов занимается использованием Linux для создания встраиваемых систем и за это время успел поработать над многими интересными проектами, например: стереоскопическая камера, «умные» весы, различные абонентские приставки и домашние маршрутизаторы и даже большой шагающий робот.

Он часто выступает на конференциях по программному обеспечению с открытым исходным кодом и по встраиваемым системам, в том числе Embedded Linux Conference, Embedded World и Android Builders' Summit. С 2002 года читает курсы и проводит семинары по встраиваемым Linux-системам, а с 2010 – по встраиванию Android. Провел сотни презентаций во многих хорошо известных компаниях. Познакомиться с его работами можно в блоге «Inner Penguin» по адресу www.2net.co.uk.

Я благодарю своего редактора Саманту Гонсалвес, которая без устали следила за ходом работы и не давала мне сбиться с пути. Также выражаю благодарность всем, кто тратил время на чтение черновых вариантов, продираясь сквозь мои путаные словеса к сути. Спасибо Бихану Уэбстеру (Behan Webster), Клаасу ван Генду (Klaas van Gend), Тиму Бэрду (Tim Bird), Роберту Бергеру (Robert Berger), Матье Дешампу (Mathieu Deschamps) и Марку Фурману (Mark Furman). И конечно, я благодарен своей супруге Ширли Симмондс, которая всегда поддерживала меня и понимала, что я никак не мог помочь ей с ремонтом дома, потому что писал книгу.

О рецензентах

Роберт Бергер с 1993 года занимается проектированием, разработкой и управлением проектами в области встраиваемых систем как с жесткими требованиями к работе в режиме реального времени, так и без оных. Начиная с начала XXI века, он использует GNU/Linux на персональных компьютерах и серверах, но в основном для целей встраивания (в автомобильные системы, средства управления производственными процессами, робототехнику, телекоммуникационное оборудование, бытовую электронику и т. д.). Он регулярно присутствует на таких международных форумах, как Embedded World, Embedded Software Engineering Congress, Embedded Systems Conference и Embedded Linux Conference, в качестве эксперта и докладчика. Специализируется главным образом на преподавании, но также оказывает консультационные услуги (на английском и немецком языках) в различных странах. В сферу профессиональной компетенции Роберта входят как совсем небольшие системы реального времени (FreeRTOS), так и комплексы с несколькими процессорами или ядрами со встроенной системой GNU/Linux (уровень пространства ядра или пользователя, драйверы устройств, интерфейсы с оборудованием, отладка, многоядерная разработка, проект Yocto Project) с упором на свободное ПО и ПО с открытым исходным кодом. Он много путешествует по свету. Является исполнительным директором и специалистом по встраиваемому ПО в компании Reliable Embedded Systems со штаб-квартирой в городе Санкт-Барбата, Австрия. Проживает с семьей в Афинах. Связаться с Робертом можно через его сайт по адресу <http://www.ReliableEmbeddedSystems.com>.

Рецензировал книгу Rudolf J. Streif «Embedded Linux Systems with the Yocto Project» (Prentice Hall Open Source Software Development Series).

Тим Бэрд работает старшим программистом в компании Sony Mobile Communications, где отвечает за модификацию ядра Linux для использования в продуктах Sony. Также является председателем архитектурной группы рабочей группы по бытовой электронике (CE Working Group) в фонде Linux Foundation. Занимается Linux свыше 20 лет. Стоял у истоков двух отраслевых объединений, связанных со встраиваемыми Linux-системами, и является основателем и бессменным руководителем конференции Embedded Linux Conference. В прошлом написал вместе с Тимом книгу «Using Caldera OpenLinux».

Матье Дешамп – основатель компании ScourGE (www.scourge.fr), предлагающей клиентам инновационные услуги в области оборудования и ПО с открытым исходным кодом. Компания занимает лидирующие позиции в сфере телекоммуникаций, мобильной связи, управления промышленными процессами и систем поддержки принятия решений.

Оказывает консультационные услуги в области исследований и разработок, преподает. С 2003 года в качестве технического руководителя принимал участие

во многих мелких и крупных проектах, так или иначе связанных с GNU/Linux, Android, разработкой встраиваемых систем и безопасностью.

Марк Фурман, автор книги «OpenVZ Essentials», в настоящее время занимает должность инженера-системотехника в компании Info-Link Technologies. В сфере ИТ работает свыше 10 лет, специализируется на Linux и других продуктах с открытым исходным кодом, а также балуется с Arduino, Python и Raspberry Pi в Knox Labs, клубе компьютерщиков-энтузиастов в округе Нокс, штат Огайо.

Клаас ван Генд окончил факультет системной инженерии и управления Эйндховенского технического университета в Нидерландах. Работал в различных компаниях, включая Philips, Siemens и Bosch, писал программное обеспечение для экспериментальных образцов принтеров, шифрования видео, автомобильных информационно-развлекательных систем, медицинского оборудования, средств автоматизации производства и навигационных систем. В 2004 году перешел в компанию MontaVista Software, являющуюся лидером на рынке встраиваемых Linux-систем. В роли системного архитектора и консультанта оказывал услуги многим европейским компаниям, помогая встраивать Linux в разработанные ими продукты.

Последние несколько лет работает преподавателем и консультантом в компании Vector Fabrics, небольшом стартапе, специализирующемся на программировании многоядерных компьютеров и динамическом анализе ПО. Преподает многоядерное программирование на C и C++, помогает заказчикам улучшить программы за счет правильного использования аппаратных ресурсов. Созданный компанией Vector Fabrics комплект инструментов Pareon помогает автоматически находить трудно воспроизводимые ошибки, в том числе гонку за данные, переполнение буфера, использование уже освобожденных областей кучи и стека, утечки памяти.

Клаас – автор более сотни журнальных статей на темы Linux (в том числе встраиваемых систем), программирования, производительности, проектирования систем и компьютерных игр. Он является соучредителем конференции Embedded Linux Conference Europe и выступал в роли ведущего разработчика в нескольких проектах с открытым исходным кодом, включая UMTSmon для сотовых сетей и игру-головоломку на темы физики The Butterfly Effect.

На досуге Клаас читает городское фэнтези или отрывается в аэроклубе Нистелроде, где летает на планерах.

Бехан Уэбстер двадцать лет работал в различных технических отраслях, включая телекоммуникации, передачу данных, оптику, встраиваемые системы и автомобильную промышленность. Писал код для самых разных устройств – от крохотных до сверхбольших. Обладает опытом программирования ядра Linux, разработки встраиваемых систем и подготовки печатных плат к производству. В настоящее время занимает должность ведущего консультанта в компании Converse in Code Inc, где занимается встраиваемыми Linux-системами, а также руководит проектом LLVMLinux и преподает в интересах фонда Linux Foundation.

Вступление

Встраиваемая система – это устройство, содержащее внутри себя компьютер, но не выглядящее как компьютер. Стиральные машины, телевизоры, принтеры, автомобили, роботы – все они управляются каким-то компьютером, а иногда и не одним. Устройства становятся все сложнее, мы ожидаем от них все большего, а значит, растут требования к управляющей ими операционной системе. И все чаще такой системой становится Linux.

Богатство возможностей Linux проистекает из модели открытых исходных текстов, поощряющей совместное владение кодом. Это означает, что инженеры с разным образованием и опытом работы, зачастую работающие в конкурирующих компаниях, могут объединить усилия для создания ядра операционной системы, поддержания его актуальности и соответствия новейшим разработкам в области оборудования. Общая кодовая база позволяет поддержать самые разные устройства – от мощнейших суперкомпьютеров до наручных часов. ОС Linux – лишь один компонент, а для создания работающей системы нужно еще много чего: от базовых инструментов типа командной оболочки до графического интерфейса пользователя, который позволяет взаимодействовать с вебом и облачными службами. Ядро Linux в сочетании с обширным набором других открытых компонентов становится основой для системы, способной исполнять различные роли.

Однако гибкость – палка о двух концах. Да, у проектировщика системы появляется широкий спектр возможных решений задачи, но вместе с ним и проблема выбора оптимального решения. Цель этой книги – подробно рассказать о том, как сконструировать встраиваемую Linux-систему из свободных программ с открытым исходным кодом, получив в результате надежный и эффективный продукт. В основу книги положен многолетний опыт работы автора в качестве консультанта и преподавателя.

Структура книги

Организационно книга устроена так же, как жизненный цикл типичного проекта встраиваемой Linux-системы. В первых шести главах изложено все, что нужно знать о подготовке проекта и устройстве системы на базе Linux, а завершается эта часть соображениями о выборе подходящей системы сборки Linux. Затем настает пора принятия ключевых решений об архитектуре и составных частях системы, в том числе флэш-памяти, драйверах устройств и системе инициализации. Следующий этап – написание приложений для работы на собранной встраиваемой платформе, этой теме посвящены две главы о процессах, потоках и управлении памятью. И наконец, в главах 12 и 13 обсуждаются отладка и оптимизация платформы. А в последней главе описана конфигурация Linux для выполнения приложений реального времени.

В главе 1 «Приступая к работе» описываются альтернативы, имеющиеся у проектировщика системы в самом начале проекта.

В главе 2 «О наборах инструментов» описаны компоненты набора инструментальных средств с упором на кросс-компиляцию. Рассказывается, где взять набор инструментов и как собрать его из исходного кода.

В главе 3 «Все о начальных загрузчиках» объяснена роль загрузчика ОС в инициализации оборудования, а в качестве примеров взяты загрузчики U-Boot и Bareboot. Описывается также дерево устройств – способ представления конфигурации оборудования, используемый во многих встраиваемых системах.

Глава 4 «Портирование и конфигурирование ядра» содержит информацию о том, как выбрать ядро Linux для встраиваемой системы и настроить его для работы с оборудованием, находящимся внутри устройства. Здесь же рассказывается о портировании Linux на новое оборудование.

В главе 5 «Построение корневой файловой системы» мы познакомимся с пользовательским пространством встраиваемой Linux-системы и представим пошаговую инструкцию по конфигурированию корневой файловой системы.

В главе 6 «Выбор системы сборки» описаны две системы сборки встраиваемых Linux-систем, цель которых – автоматизация описанных ранее шагов. На этом завершается первая часть книги.

Глава 7 «Выбор стратегии хранения» посвящена обсуждению проблем, связанных с управлением флэш-памятью, в том числе неуправляемых микросхем флэш-памяти и карт типа ММС и eMMC для встраиваемых систем. Описаны файловые системы для каждой технологии. Рассматривается также вопрос об обновлении прошивки устройства пользователем.

В главе 8 «Введение в драйверы устройств» описывается, как находящийся в ядре драйвер устройства взаимодействует с оборудованием. Приведены примеры простых драйверов. Описаны также различные способы вызова драйвера из пользовательского адресного пространства.

В главе 9 «Инициализация системы – программа init» описано, как запускается первая программа, работающая в пользовательском пространстве, – `init`, и как она запускает все остальные программы в системе. Мы рассматриваем три варианта программы `init`, подходящие для встраиваемых систем разного типа: от самой простой, `BusyBox init`, до `systemd`.

Глава 10 «Процессы и потоки» посвящена рассмотрению встраиваемых систем с точки зрения прикладного программиста. В ней речь пойдет о процессах, потоках, межпроцессном взаимодействии и политиках планирования.

В главе 11 «Управление памятью» обсуждаются идеи, лежащие в основе механизма виртуальной памяти и отображения адресного пространства. Рассказано также о том, как узнать, какие части памяти используются, и обнаружить утечки памяти.

Глава 12 «Отладка в GDB» посвящена использованию отладчика GDB для интерактивной отладки кода, работающего в ядре и в пользовательском пространстве. Описан также отладчик ядра `kdb`.

В главе 13 «Профилирование и трассировка» рассматриваются методы измерения производительности системы, начиная с получения профиля работы системы в целом и заканчивая выявлением узких мест, приводящих к снижению производительности. Здесь же описана программа Valgrind, которая проверяет, правильно ли приложение синхронизирует потоки и управляет памятью.

В главе 14 «Программирование в режиме реального времени» вы найдете подробное руководство по созданию программ реального времени в Linux, включая настройку ядра и наложение на него заплат реального времени, а также описание инструментов для измерения задержек. Здесь же приводятся сведения о том, как уменьшить число страничных прерываний путем блокировки памяти.

Что необходимо для чтения этой книги

В книге используется только программное обеспечение с открытым исходным кодом, в большинстве случаев – последние версии, доступные на момент написания книги. Я старался описывать основные возможности, не привязываясь к конкретной версии, но некоторые команды могут не работать с более поздними версиями – от этого никуда не денешься. Надеюсь, впрочем, что сопроводительные описания достаточно информативны, так что вам не составит труда применить те же принципы к последующим версиям пакета.

В создании любой встраиваемой системы участвуют две системы: исходная – в которой кросс-компилируется программный код, и целевая – в которой этот код будет работать. В качестве исходной системы я использовал Ubuntu 14.04, но годится (возможно, с минимальной модификацией) почти любой дистрибутив Linux. Я вынужден был выбрать какую-то целевую систему для представления встраиваемой. В книге рассматриваются два варианта: BeagleBone Black и QEMU – эмулятор системы с процессором ARM. Второй вариант означает, что примеры можно выполнять, не тратясь на оборудование для реального целевого устройства. При этом ничто не мешает попробовать примеры на различных целевых платформах, изменив такие детали, как имена устройств и распределение памяти.

Версии основных пакетов для целевой системы: U-Boot 2015.07, Linux 4.1, Yocto Project 1.8 «Fido» и Buildroot 2015.08.

Предполагаемая аудитория

Эта книга рассчитана в первую очередь на разработчиков Linux и системных программистов, которые уже знакомы со встраиваемыми системами и хотят узнать, как создавать лучшие в своем классе устройства. Требуется понимание основ программирования на языке C и опыт системного программирования.

Графические выделения

В этой книге для выделения семантически различной информации применяются различные стили. Ниже приведены примеры стилей с пояснениями.

Фрагменты, имена функций, таблиц базы данных, папок и файлов, URL-адреса, адреса в Twitter и данные, вводимые пользователем, выглядят следующим образом: «Мы могли бы воспользоваться функциями потокового ввода-вывода `foopen(3)`, `fread(3)` и `fclose(3)`».

Отдельно стоящие фрагменты кода набраны так:

```
static struct mtd_partition omap3beagle_nand_partitions[] = {
    /* Размеры разделов задаются в терминах размера блока NAND-памяти */
    {
        .name = "X-Loader",
        .offset = 0,
        .size = 4 * NAND_BLOCK_SIZE,
        .mask_flags = MTD_WRITEABLE, /* только для чтения */
    }
}
```

Чтобы привлечь внимание к части кода, строки или отдельные слова выделяются полужирным шрифтом:

```
static struct mtd_partition omap3beagle_nand_partitions[] = {
    /* Размеры разделов задаются в терминах размера блока NAND-памяти */
    {
        .name = "X-Loader",
        .offset = 0,
        .size = 4 * NAND_BLOCK_SIZE,
        .mask_flags = MTD_WRITEABLE, /* только для чтения */
    }
}
```

Текст, который вводится на консоли или выводится на консоль, напечатан следующим образом:

```
# flash_erase -j /dev/mtd6 0 0
# nandwrite /dev/mtd6 rootfs-sum.jffs2
```

Новые термины и важные слова набраны полужирным шрифтом. Также выделяются элементы интерфейса, например пункты меню и поля в диалоговых окнах: «Во второй строке на консоль выводится сообщение **Please press Enter to activate this console**».



Предупреждения и важные замечания оформлены так.



Советы и рекомендации выглядят так.

Отзывы

Мы всегда рады отзывам читателей. Расскажите нам, что вы думаете об этой книге – что вам понравилось или, быть может, не понравилось. Читательские отзывы

важны для нас, так как помогают выпускать книги, из которых вы черпаете максимум полезного для себя.

Чтобы отправить обычный отзыв, просто пошлите письмо на адрес feedback@packtpub.com, указав название книги в качестве темы.

Если вы являетесь специалистом в некоторой области и хотели бы стать автором или соавтором книги, познакомьтесь с инструкциями для авторов по адресу www.packtpub.com/authors.

Поддержка клиентов

Счастливым обладателям книг Packt мы можем предложить ряд услуг, которые позволят извлечь из приобретения максимум пользы.

Загрузка кода примеров

Вы можете скачать код примеров ко всем приобретенным вами книгам издательства Packt в своем личном кабинете на сайте <http://www.PacktPub.com>. Если книга была куплена в другом месте, зайдите на страницу <http://www.PacktPub.com/support>, зарегистрируйтесь, и мы отправим файлы по электронной почте.

Опечатки

Мы проверяли содержимое книги очень тщательно, но какие-то ошибки все же могли проскользнуть. Если вы найдете в нашей книге ошибку, в тексте или в коде, пожалуйста, сообщите нам о ней. Так вы избавите других читателей от разочарования и поможете нам сделать следующие издания книги лучше. При обнаружении опечатки просьба зайти на страницу <http://www.packtpub.com/submit-errata>, выбрать книгу, щелкнуть по ссылке **Errata Submission Form** и ввести информацию об опечатке. Проверив ваше сообщение, мы поместим информацию об опечатке на нашем сайте или добавим ее в список замеченных опечаток в разделе **Errata** для данной книги.

Список подтвержденных опечаток можно просмотреть, введя название книги в поле поиска на странице <https://www.packtpub.com/books/content/support>. Информация появится в разделе **Errata**.

Нарушение авторских прав

Незаконное размещение защищенного авторским правом материала в Интернете – проблема для всех носителей информации. В издательстве Packt мы относимся к защите прав интеллектуальной собственности и лицензированию очень серьезно. Если вы обнаружите незаконные копии наших изданий в любой форме в Интернете, пожалуйста, незамедлительно сообщите нам адрес или название веб-сайта, чтобы мы могли предпринять соответствующие меры.

Просим отправить ссылку на вызывающий подозрение в пиратстве материал по адресу copyright@packtpub.com.

Мы будем признательны за помощь в защите прав наших авторов и содействие в наших стараниях предоставлять читателям полезные сведения.

Вопросы

Если вас смущает что-то в этой книге, вы можете связаться с нами по адресу questions@packtpub.com, и мы сделаем все возможное для решения проблемы.

Приступая к работе

Итак, вы приступаете к работе над очередным проектом, и на этот раз он будет основан на Linux. О чем следует подумать, прежде чем бежать к клавиатуре? Начнем с общего обзора встраиваемых Linux-систем, объясним, почему они так популярны, что подразумевают лицензии на ПО с открытым исходным кодом и какого рода оборудование необходимо для работы Linux.

Linux стала рассматриваться в качестве кандидата на роль управляющей ОС для встраиваемых устройств где-то в 1999 году. Именно тогда компания Axis (www.axis.com) выпустила свою первую сетевую камеру под управлением Linux, а компания TiVo (www.tivo.com) – первый видеоманитон. С тех пор популярность Linux постоянно росла, и сегодня это основная операционная система для многих классов изделий. На момент написания этой книги, в 2015 году, под управлением Linux работают примерно два миллиарда устройств. Сюда входит неисчислимая рать смартфонов на платформе Android с ядром Linux, сотни миллионов абонентских приставок, «умных» телевизоров (Smart TV) и маршрутизаторов Wi-Fi, а также множество других устройств, производимых меньшими тиражами, например: приборы автомобильной диагностики, весы, промышленные устройства и медицинские мониторы.

Так почему же в вашем телевизоре работает Linux? На первый взгляд, от телевизора требуется всего лишь отобразить на экране поток видеоданных – не так уж сложно. Зачем для этого навороченная операционная система типа Linux?

Простой ответ дает закон Мура: Гордон Мур, один из основателей компании Intel, в 1965 г. заметил, что плотность компонентов на кристалле микросхемы удваивается примерно каждые два года. Это относится не только к настольным компьютерам, ноутбукам и серверам, но и к устройствам, которыми мы пользуемся в повседневной жизни. Сердцем большинства встраиваемых устройств является интегральная микросхема, содержащая одно или несколько процессорных ядер и интерфейсы с оперативной памятью, массовым запоминающим устройством и разнообразными периферийными устройствами. Все это называется «системой на кристалле» (System on Chip, или SoC), и сложность таких систем растет в соответствии с законом Мура. К типичной SoC-системе прилагается справочное техническое руководство, насчитывающее тысячи страниц. Ваш телевизор не просто отображает видеосигнал, как старый добрый аналоговый приемник.

Поток данных представлен в цифровой форме и иногда зашифрован, его необходимо обработать, чтобы получить изображение. Ваш телевизор подключен к Интернету (или будет подключен в ближайшем будущем). Он может получать контент от смартфонов, планшетов и домашних медиа-серверов. На нем можно играть в игры. И так далее и тому подобное. Для управления всеми этими сложными функциями нужна полноценная операционная система.

Ниже перечислено несколько причин для выбора на эту роль Linux.

- В Linux уже имеются необходимая функциональность: хороший планировщик задач, хороший сетевой стек, поддержка USB, Wi-Fi, Bluetooth, многих запоминающих устройств, мультимедийных устройств и т. д. В общем, против любого пункта стоит галочка.
- Linux портирована на многие процессорные архитектуры, в том числе часто применяемые в системах на кристалле: ARM, MIPS, x86 и PowerPC.
- Исходный код Linux открыт, так что в него можно внести необходимые вам изменения. Вы или кто-то, работающий по вашему поручению, может создать пакет программ для поддержки конкретной SoC-платы или устройства. Можно добавить протоколы, функции и технологии, которых нет в базовом исходном коде. Или исключить ненужные возможности, чтобы уменьшить требования к оперативной и внешней памяти. Linux – гибкая система.
- Вокруг Linux сложилось активное сообщество, а если говорить о ядре, то даже очень активное. Новая версия ядра выходит раз в 10–12 недель, причем авторами кода являются порядка 1000 разработчиков. Такая активность означает, что Linux всегда актуальна и поддерживает все современное оборудование, протоколы и стандарты.
- Лицензия на ПО с открытым исходным кодом гарантирует доступ к исходному коду. Вы не привязаны к конкретному поставщику.

В силу этих причин Linux – идеальный выбор для создания сложных систем. Но есть и несколько подводных камней, о которых нельзя не упомянуть. В придачу к быстрому процессу разработки и децентрализованной структуре управления полагается нагрузка: вы должны приложить усилия, чтобы разобраться в том, как все это устроено, и продолжать учиться по мере дальнейшего развития. Надеюсь, эта книга поможет вам справиться.

Выбор правильной операционной системы

Подходит ли Linux для вашего проекта? Linux стоит рассматривать, если решаемая задача оправдывает сложность системы. Особенно это относится к задачам, где важны способность к выходу в сеть, надежность и наличие развитых интерфейсов с пользователем. Однако не всякую проблему можно решить с помощью Linux, и ниже приведены вопросы, которые вы должны задать себе, прежде чем принимать решение.

- Подходит ли ваше оборудование для работы с Linux? По сравнению с традиционными операционными системами реального времени (ОСРВ) типа

VxWorks, Linux значительно требовательнее к ресурсам. Ей нужны, по меньшей мере, 32-разрядный процессор и гораздо больше памяти. Я еще вернусь к этому вопросу в разделе о типичных требованиях к оборудованию.

- Обладаете ли вы необходимыми знаниями и умениями? На ранних стадиях проекта, когда идет подготовка печатной платы к производству, потребуются детальное знание Linux и ее взаимодействие с вашим оборудованием. А на этапе отладки и оптимизации приложения вам придется интерпретировать результаты, и нужно знать, как это делается. Если внутри компании нет специалистов нужной квалификации, то, возможно, имеет смысл поручить часть работы внешним организациям. Но, конечно, чтение этой книги выручит!
- Должна ли ваша система работать в режиме реального времени? Linux способна справиться со многими задачами реального времени, если уделять внимание некоторым деталям, о которых я расскажу в главе 14.

Подойдите к этим вопросам со всей серьезностью. Быть может, стоит поискать похожие продукты, в которых используется Linux, и поинтересоваться, как они сделаны. Следуйте передовым образцам.

Игроки

Откуда берутся программы с открытым исходным кодом? Кто их пишет? И как они соотносятся с основными компонентами разработки встраиваемых систем – набором инструментов, начальным загрузчиком, ядром и основными утилитами, находящимися в корневой файловой системе?

Перечислим основных игроков.

- Сообщество, сложившееся вокруг ПО с открытым исходным кодом. Это и есть движущая сила, стоящая за тем программным обеспечением, которое вы собираетесь использовать. Сообщество – это неформальное объединение разработчиков, многие из которых тем или иным способом получают вознаграждение за свой труд, например, от некоммерческой организации, академического учреждения или коммерческой компании. Они совместно работают во имя достижения целей различных проектов. Сообществ много – больших и маленьких. В этой книге упоминаются сообщества вокруг самой ОС Linux, проектов U-Boot, BusyBox, Buildroot, Yocto Project и многих проектов под эгидой GNU.
- Архитекторы процессоров. Это организации, занимающиеся проектированием используемых нами процессоров. Основные из них: ARM/Linaro (SoC-системы на базе процессоров ARM), Intel (платформы x86 и x86_64), Imagination Technologies (MIPS) и Freescale/IBM (PowerPC). Они реализуют основные архитектуры процессоров или, по крайней мере, оказывают влияние на поддержку.
- Поставщики систем на кристалле (Atmel, Broadcom, Freescale, Intel, Qualcomm, TI и многие другие). Они получают ядро и набор инструментов от архи-

текторов процессоров и модифицируют их для поддержки своих микросхем. Они же разрабатывают эталонные платы: конструкции, которые на следующем уровне используются для создания макетных плат и готовых изделий.

- Поставщики плат и производители комплектного оборудования (ОЕМ) – организации, которые получают типовой вариант конструкции от поставщиков систем на кристалле и встраивают его в конкретное изделие, например абонентскую приставку или камеру, либо создают макетные платы более общего назначения, как, например, компании Avantech или Kontron. Важным классом подобных изделий являются дешевые макетные платы, например BeagleBoard/BeagleBone и Raspberry Pi, вокруг которых сложились свои экосистемы программных и аппаратных дополнений.

Ваш проект обычно оказывается конечным звеном такой цепочки, т. е. свободы выбора компонентов вы лишены. Не получится просто взять последнюю версию ядра с сайта `kernel.org` (разве что в редчайших случаях), потому что она не поддерживает используемую вами микросхему или плату.

Это общая проблема, присущая разработке встраиваемых систем. В идеале разработчики, занятые в каждом звене, должны передавать сделанные ими изменения на следующий уровень, но они этого не делают. Не редкость встретить ядро с тысячами заплат, которые не пошли дальше. К тому же поставщики SoC-систем склонны активно разрабатывать открытый код только для последних версий своих микросхем, а значит, поддержка любой микросхемы после примерно двух лет замораживается, и выпуск обновлений прекращается.

В результате большинство встраиваемых систем основано на устаревших версиях ПО. Для них не выходят обновления, повышающие безопасность и производительность, им недоступны функции, имеющиеся в более поздних версиях. Уязвимости типа Heartbleed (из-за ошибки в библиотеках OpenSSL) или Shellshock (из-за ошибки в оболочке bash) так и остаются незакрытыми. Я еще вернусь к этому вопросу при обсуждении темы безопасности в этой главе.

Что вы можете с этим поделать? Во-первых, задать вопросы поставщикам: какова их стратегия обновления, как часто они пересматривают версии ядра, какую версию ядра используют сейчас и какую использовали перед этим? Некоторые поставщики добились заметного прогресса в этом отношении, поэтому стоит предпочесть их продукцию.

Во-вторых, вы можете предпринять некоторые шаги по обеспечению собственной автономности. В этой книге будут подробно рассмотрены зависимости и показано, что можно сделать в этом отношении. Не стоит слепо использовать пакет, предложенный производителем SoC-системы или платы, даже не посмотрев, какие есть альтернативы.

Жизненный цикл проекта

Эта книга состоит из четырех частей, соответствующих этапам работы над проектом. Этапы не обязательно следуют один за другим. Обычно они перекрываются,

иногда приходится возвращаться назад и пересматривать ранее принятые решения. Тем не менее они дают представление о типичной деятельности разработчика проекта.

- Главы 1-6, посвященные элементам встраиваемых Linux-систем, помогут организовать среду разработки и создать платформу для работы на следующих этапах. Часто этот этап называют «подготовкой платы к производству» (board bring-up).
- Главы 7–9 о выборе архитектуры и конструкции системы позволят взглянуть на некоторые проектные решения, относящиеся к хранению программ и данных, к разделению работы между драйверами устройств в ядре и приложениями и к инициализации системы.
- Главы 10 и 11 посвящены встраиваемым приложениям, в них рассказано, как эффективно использовать модель процессов и потоков в Linux и как управлять памятью в условиях ограниченности ресурсов.
- В главах 12 и 13, посвященных отладке и оптимизации, описаны методы трассировки, профилирования и отладки кода приложения и ядра.

Пятая часть, состоящая из главы 14, стоит особняком и посвящена системам реального времени – небольшому, но важному классу встраиваемых систем. Учет поведения в реальном времени оказывает влияние на все четыре основных этапа.

Четыре составные части встраиваемой Linux-системы

Любой проект начинается с получения, модификации под свои нужды и развертывания четырех элементов: набора инструментов, начального загрузчика, ядра и корневой файловой системы. Этой теме посвящена первая часть книги.

- **Набор инструментов:** состоит из компилятора и прочих инструментальных средств, необходимых для создания программы, управляющей целевым устройством. От набора инструментов зависит все остальное.
- **Начальный загрузчик** необходим для инициализации платы, загрузки и инициализации ядра Linux.
- **Ядро:** это сердце системы, оно управляет всеми ресурсами и осуществляет взаимодействие с оборудованием.
- **Корневая файловая система** содержит библиотеки и программы, исполняемые после завершения инициализации ядра.

Есть еще и пятый, не упомянутый здесь элемент. Это набор программ, составляющих ваше встраиваемое приложение, благодаря которому устройство выполняет свои функции, будь то взвешивание бакалейных товаров, показ фильмов, управление роботом или полет дрона.

Как правило, все или некоторые из этих элементов предлагаются в виде пакета при покупке нужной вам SoC-системы или платы. Но по вышеупомянутым причинам это не всегда оптимальный выбор. Материал первых шести глав поможет принять правильное решение. Мы также познакомимся с двумя инструментами автоматизации: Buildroot и Yocto Project.

Программное обеспечение с открытым исходным кодом

Компоненты встраиваемой Linux-системы – программы с открытым исходным кодом, поэтому самое время понять, что это означает, почему такой подход вообще работает и как отражается на создаваемом с помощью таких программ устройстве, которое зачастую защищено правом собственности.

Лицензии

Говоря об открытом исходном коде, часто употребляют слово «free»¹. Люди, не знакомые с предметом, обычно предполагают, что речь идет о бесплатности, и лицензия действительно гарантирует, что ПО можно использовать для разработки и внедрения систем и никому ничего не платить. Однако более важно другое значение слова «free» – свободный, поскольку вы свободны в своем праве получить исходный код, модифицировать его, как считаете нужным, и распространять в составе других систем. Эти лицензии дают такое право. Сравните с условно-бесплатными лицензиями, которые разрешают бесплатно копировать двоичный код, но не дают доступа к исходному, или с лицензиями, разрешающими бесплатно использовать ПО только при определенных условиях, например для личного пользования, но запрещают использование в коммерческих целях. Ни то, ни другое нельзя назвать «открытым исходным кодом».

Чтобы вам было проще понять последствия работы с лицензиями на ПО с открытым исходным кодом, я дам несколько пояснений, но имейте в виду, что я инженер, а не юрист. Ниже изложено мое личное понимание лицензий.

Лицензии на ПО с открытым исходным кодом можно отнести к одному из двух классов: **GPL (General Public License** – генеральная общедоступная лицензия), предоставляемая Фондом свободного ПО (Free Software Foundation), и либеральные лицензии, производные от лицензий **BSD (Berkeley Software Distribution)**, фонда Apache Foundation и др.

Либеральная лицензия по существу означает, что вы вправе модифицировать исходный код и использовать его в своих системах при условии неизменности условий самой лицензии. Иными словами, при этом единственном ограничении вы можете делать с кодом все, что угодно, в том числе встраивать его в системы, защищенные правом собственности.

Лицензии GPL похожи, но включают оговорку, в соответствии с которой вы обязаны передать право на получение и модификацию ПО конечным пользователям вашего продукта. Иными словами, вы передаете свой исходный код в общее пользование. Один из вариантов – сделать его доступным для всех желающих, выложив на публичный сервер. Другой – предоставлять только вашим конечным пользователям, включив письменное обязательство предоставить код по запросу. Лицензия GPL идет еще дальше и запрещает включать распространяемый на ее

¹ В английском языке free означает как «бесплатный», так и «свободный». – *Прим. перев.*

условиях код в закрытые программы. Любое включение такого кода автоматически означает, что ко всей программе применяются условия GPL. Иначе говоря, в одной программе не может встречаться закрытый код и код, распространяемый на условиях GPL.

А как насчет библиотек? Если библиотека распространяется на условиях GPL, то любая скомпонованная с ней программа должна распространяться на тех же условиях. Однако большинство библиотек распространяется на условиях лицензии **LGPL (Lesser General Public License)**, которая разрешает компоновать библиотеку с закрытой программой.

Все вышесказанное относится к лицензиям GPL v2 и LGPL v2.1. Следует сказать также о последних версиях GPL v3 и LGPL v3. Они противоречивы, и признаюсь, что не вполне понимаю, какие последствия они влекут. Но идея была в том, чтобы гарантировать следующее положение: любой компонент системы, распространяемый по лицензии GPL v3 и LGPL v3, должен допускать замену конечным пользователем. И это, конечно, согласуется с духом ПО с открытым исходным кодом. Но есть и проблемы. Некоторые устройства, управляемые Linux, применяются для получения доступа к информации согласно с уровнем подписки или иным ограничением, а замена критической части программы может снять ограничение. Типичный пример – абонентские приставки. Есть также проблемы, связанные с безопасностью. Если владелец устройства имеет доступ к системному коду, то его может получить и незваный гость. Для защиты часто используют образы ядра, подписанные уполномоченным органом или поставщиком, так что несанкционированное изменение невозможно. Это следует считать покушением на мое право модифицировать собственное устройство? Есть разные мнения.



В этом споре часто упоминается абонентская приставка TiVo. В ней используется ядро Linux, распространяемое по лицензии GPL v2. Компания TiVo раскрывает код своей версии ядра – в полном соответствии с лицензией. Но у TiVo имеется также начальный загрузчик, который соглашается загружать только подписанный ей же двоичный код ядра. Следовательно, мы можем собрать модифицированное ядро для приставки TiVo, но не сумеем загрузить его в оборудование. Позиция Фонда открытого ПО заключается в том, что это не в духе ПО с открытым исходным кодом, а для самой процедуры используется термин «tivoизация». Версии GPL v3 и LGPL v3 специально разработаны, чтобы запретить такую ситуацию. Разработчики некоторых проектов и, в частности, ядра Linux не торопятся принимать лицензии третьей версии из-за ограничений на производителей оборудования.

Оборудование для встраиваемых Linux-систем

На что обращать внимание при проектировании или выборе оборудования для проекта встраиваемой Linux-системы?

Прежде всего на архитектуру процессора, поддерживаемую ядром, – если, конечно, вы не собираетесь самостоятельно разработать новую архитектуру! В исходном коде Linux 4.1 есть 30 архитектур, представленных отдельными подката-

логами каталога arch/. Среди них имеются 32- и 64-разрядные архитектуры, по большей части с блоком управления памятью (MMU), хотя это необязательно. Во встраиваемых устройствах чаще всего встречаются архитектуры ARM, MIPS, PowerPC и X86, каждая из которых существует в 32- и 64-разрядных вариантах, и все они включают блок управления памятью.

В этой книге в основном подразумеваются процессоры такого типа. Есть также группа устройств, не оснащенных MMU и управляемых подмножеством Linux, называемым «Linux для микроконтроллеров», или uClinux. В эту группу входят архитектуры процессоров ARC, Blackfin, Microblaze и Nios. В нескольких местах я упоминаю uClinux, но в детали не вхожу, потому что это слишком специальная тема.

Во-вторых, нужно иметь достаточный объем оперативной памяти (ОЗУ). 16 МиБ – неплохой минимум, хотя, в принципе, для работы Linux достаточно и половины. Можно запустить Linux даже в 4 МиБ, если вы готовы потратить усилия на оптимизацию каждой части системы. Можно пойти и дальше, но рано или поздно наступает момент, когда систему уже нельзя назвать Linux.

В-третьих, встает вопрос об энергонезависимой памяти, обычно флэш-памяти. 8 МиБ достаточно для простого устройства типа веб-камеры или простенького маршрутизатора. Как и в случае ОЗУ, при желании можно построить работоспособную Linux-систему с меньшим объемом внешней памяти, но чем он меньше, тем труднее становится задача. Linux прекрасно поддерживает флэш-память, в том числе приборы, выполненные по технологиям NOR и NAND, и управляемую флэш-память в форме SD-карт, микросхем eMMC, флэш-памяти с интерфейсом USB и т. д.

В-четвертых, очень полезен порт для отладки, обычно это порт последовательного интерфейса RS-232. Оснащать им готовое изделие не обязательно, но на этапе подготовки платы к производству он заметно ускоряет разработку и отладку.

В-пятых, если вы начинаете с чистого листа, то понадобятся какие-то средства загрузки программы. Несколько лет назад для этой цели платы оснащались интерфейсом JTAG, но современные системы на кристалле умеют читать код начальной загрузки прямо со съемного запоминающего устройства, в частности карты SD или microSD, или из последовательного интерфейса типа RS-232 или USB.

Помимо этих базовых вещей, существуют интерфейсы к специальному оборудованию, необходимому для работы устройства. В стандартную комплектацию Linux входят открытые драйверы для тысяч разных устройств, и существуют также драйверы (разного качества), написанные производителями SoC-систем и сторонних микросхем, которые тоже можно включить в проект. Однако не забывайте о моих комментариях по поводу добросовестности и возможностей производителей. Любой разработчик встраиваемых устройств тратит уйму времени на оценку и адаптацию стороннего кода, если он доступен, и на консультации с производителем в противном случае. Наконец, вам предстоит написать драйверы для уникальных интерфейсов разрабатываемого устройства или подрядить кого-то для выполнения этой работы.

Используемое оборудование

Приведенные в книге примеры носят общий характер, но чтобы их можно было выполнить, я был вынужден выбрать какое-то конкретное устройство. В качестве таковых я использую BeagleBone Black и QEMU. Первое – широко распространенная дешевая макетная плата, которую можно использовать для разработки серьезных встраиваемых систем. Второе – эмулятор, применяемый для создания широкого спектра типичного встраиваемого оборудования. Был соблазн ограничиться только QEMU, но, как и любой эмулятор, он все же не полностью имитирует реальное устройство. Работая с BeagleBone, вы получаете удовольствие от взаимодействия с настоящим оборудованием и видите, как мигают лампочки. Был также соблазн взять что-то поновее платы BeagleBone Black, которая была выпущена несколько лет назад, но я полагаю, что заслуженная популярность обеспечила ей долгую жизнь, так что еще сколько-то лет она никуда не денется.

Как бы то ни было, советую выполнить как можно больше примеров на одной из этих платформ или на любой другой, которая окажется под рукой.

Плата BeagleBone Black

BeagleBone и более поздняя версия BeagleBone Black – это макетные платы размером с кредитную карту производства компании Circuitco LLC, проходящие по ряду открытого аппаратного обеспечения. Авторитетный источник информации о них – сайт www.beagleboard.org. Вот их основные технические характеристики:

- TI AM335x 1 ГГц ARM® Cortex-A8 Sitara SoC;
- ОЗУ 512 МиБ DDR3;
- флэш-память на плате: 8-разрядная eMMC объемом 2 или 4 ГиБ;
- последовательный порт для разработки и отладки;
- разъем для карты microSD, которую можно использовать как загрузочное устройство;
- порт хоста/устройства mini-USB OTG, который можно использовать для подачи питания плате;
- полноразмерный порт хоста USB 2.0;
- порт Ethernet 10/100;
- порт HDMI для аудио- и видеовыхода.

Кроме того, имеются два 46-контактных разъема для подключения карт расширения, каковых существует великое множество, – они позволяют приспособить плату для выполнения различных функций. Впрочем, в этой книге карты расширения не используются.

Помимо самой платы, нам понадобятся:

- переходник между mini-USB и полноразмерным USB (прилагается к плате) для подачи питания, если только у вас нет аксессуара, указанного в конце этого списка;
- кабель RS-232, который можно подключить к 6-контактному элементу TTL с напряжением питания 3,3 В, имеющемуся на плате. На сайте BeagleBoard приведены ссылки на совместимые кабели;

- карта microSD и средства записи на нее с настольного ПК или ноутбука, на котором ведется разработка, она будет нужна для загрузки программы в память платы;
- Ethernet-кабель, поскольку для некоторых примеров нужен доступ к сети;
- необязательно, но рекомендуется 5-вольтовый источник питания с силой тока не менее 1 А.

QEMU

QEMU – это эмулятор оборудования. Он поставляется в разных вариантах, предназначенных для эмуляции той или иной архитектуры процессора и ряда плат на основе этой архитектуры, например:

- `qemu-system-arm`: ARM;
- `qemu-system-mips`: MIPS;
- `qemu-system-ppc`: PowerPC;
- `qemu-system-x86`: x86 and x86_64.

Для каждой архитектуры QEMU эмулирует различное оборудование, перечень которого можно увидеть, запустив эмулятор с параметром `-machine help`. Каждая «машина» эмулирует большую часть оборудования, обычно присутствующего на данной плате. Существует возможность связать оборудование с локальными ресурсами, скажем, использовать локальный файл в качестве эмулируемого дискового накопителя. Вот конкретный пример:

```
$ qemu-system-arm -machine vexpress-a9 -m 256M -drive file=rootfs.ext4,sd
-net nic -net use -kernel zImage -dtb vexpress-v2p-ca9.dtb
-append "console=ttyAMA0,115200 root=/dev/mmcblk0" - serial stdio
-net nic,model=lan9118 -net tap,ifname=tap0
```

Опишем параметры этой команды:

- `-machine vexpress-a9`: эмулировать макетную плату ARM Versatile Express с процессором Cortex A-9;
- `-m 256M`: наделить ее ОЗУ объемом 256 МиБ;
- `-drive file=rootfs.ext4,sd`: связать интерфейс `sd` с локальным файлом `rootfs.ext4` (который содержит образ файловой системы);
- `-kernel zImage`: загрузить ядро Linux из локального файла с именем `zImage`;
- `-dtb vexpress-v2p-ca9.dtb`: загрузить дерево устройств из локального файла `vexpress-v2p-ca9.dtb`;
- `-append "..."`: использовать эту строку как строку для инициализации ядра;
- `-serial stdio`: присоединить последовательный порт к терминалу, с которого запускался QEMU, обычно это делается для того, чтобы можно было войти в эмулируемую машину с последовательной консолью;
- `-net nic,model=lan9118`: создать сетевой интерфейс;
- `-net tap,ifname=tap0`: присоединить сетевой интерфейс к интерфейсу виртуальной сети `tap0`.

Чтобы сконфигурировать часть сети, относящуюся к хосту, понадобится утилита `tunctl` из проекта **User Mode Linux (UML)**; в дистрибутивах Debian и Ubuntu

пакет называется `uml-utilities`. С помощью этой утилиты виртуальная сеть создается следующим образом:

```
$ sudo tuncctl -u $(whoami) -t tap0
```

Эта команда создает сетевой интерфейс `tap0`, который подключается к сетевому контроллеру эмулируемой QEMU машины. Затем `tap0` настраивается точно так же, как любой интерфейс.

Все параметры будут подробно описаны в последующих главах. В большинстве примеров используется плата Versatile Express, но взять другую машину или архитектуру столь же просто.

Используемое программное обеспечение

Все используемые инструменты разработки, целевая операционная система и приложения – программы с открытым исходным кодом. Предполагается, что разработка ведется в Linux. Все команды, выполняемые хостом, тестировались в системе Ubuntu 14.04, но должен подойти любой современный дистрибутив Linux.

Резюме

Встраиваемое оборудование со временем будет становиться все сложнее – в полном соответствии с законом Мура. Linux обладает достаточными возможностями и гибкостью для эффективной работы с оборудованием.

Linux – лишь один компонент ПО с открытым исходным кодом. Помимо него, для создания готового изделия нужно еще много чего. Поскольку весь код свободно доступен, свой вклад могут вносить многие люди и организации. Однако из-за многообразия встраиваемых платформ и высокой скорости разработки образуются изолированные островки ПО, для которых совместное владение организовано не настолько эффективно, как могло бы быть. Во многих случаях вы попадаете в зависимость от этого ПО, особенно если ядро Linux поставляется изготовителем SoC-системы или печатной платы. К набору инструментов это относится в меньшей степени. Некоторые изготовители SoC-систем более ответственно относятся к публикации внесенных изменений, тогда их сопровождение упрощается.

По счастью, имеются эффективные инструменты, позволяющие разрабатывать и сопровождать программное обеспечение устройства. Так, Buildroot идеален для небольших систем, а Yocto Project – для более крупных.

Прежде чем переходить к описанию этих средств сборки, я рассмотрю четыре элемента встраиваемой Linux-системы, которые встречаются во всех проектах, как бы они ни создавались. Следующая глава посвящена первому элементу – набору инструментов, необходимому, чтобы откомпилировать код для целевой платформы.