Оглавление

Старт Linux для ARM

http://www.simtec.co.uk/products/SWLINUX/files/booting_article.pdf

Booting ARM Linux

Vincent Sanders <vince@arm.linux.org.uk>
Проверка и советы, большие куски ядра, всё вокруг хорошего парня: Russell King

Проверка, советы и многочисленные пояснения: Nicolas Pitre

Проверка и советы: Erik Mouw, Zwane Mwaikambo, Jeff Sutherland, Ralph Siemsen, Daniel Silverstone, Martin Michlmayr, Michael Stevens, Lesley Mitchell, Matthew Richardson

Проверка и справочная информация (смотрите библиографию): Wookey

2004-06-04

История версий
Revision 1.00 10th May 2004 VRS

Начальный вариант.
Revision 1.10 4th June 2004 VRS

Обновлён пример кода, чтобы он стал более полным.
Улучшение формулировок в разных местах, изменения, предложенные Nicolas Pitre.
Обновлён Раздел 2, "Другие загрузчики".
Обновлены благодарности.

Краткие сведения

Этот документ определяет в ясных кратких терминах, с руководством по реализации и примерами, требования и процедуры для загрузчика, чтобы запустить ядро Linux на ARM.

1. Об этом документе

Этот документ описывает "новую" процедуру загрузки, которую используют все версии ядра, начиная с 2.4.18. Устаревший метод "struct" использоваться ***не*** должен.

Этот документ содержит информацию из самых разнообразных источников (смотрите Библиографию) и от других авторов, вам предлагается ознакомиться с этими источниками для получения дополнительной информации, прежде чем задавать вопросы поддержке или в рассылке ARM Linux. Большинство из этих областей неоднократно описывались в прошлом и будут, вероятно, игнорироваться, если вы имеете базовый опыт.

Дополнительно следует отметить, что приводимое в этом документе руководство полагает, что не должно быть никакой необходимости исполнителем понимать все нюансы сборки, которая запускает ядро. Опыт показал, что большинство проблем запуска вряд ли будет связано с этим кодом, вышеупомянутый код довольно хитрый и вряд ли даст какое-либо представление о проблеме.

2. Другие загрузчики

Прежде чем приступить к написанию нового загрузчика, разработчик должен рассмотреть, не является ли подходящим один из существующих загрузчиков. Есть примеры загрузчиков в большинстве областей, начиная от простых погрузчиков под лицензией GPL до сильно развитых коммерческих предложений. Здесь приводится краткий список, но документы в Библиографии предлагают больше решений.

Таблица 1, загрузчики

Название	URL	Описание
Blob	Blob bootloader [http://www.sf.net/projects/blob/]	Загрузчик под лицензией GPL для платформ SA11x0 (StrongARM).
Bootldr	Bootldr [http://www.handhelds.org/sources.html]	Доступны версии под GPL и не-GPL, главным образом используется в карманных устройствах.
Redboot	Redboot [http://sources.redhat.com/redboot/]	Загрузчик Redhat, выпущенный под их лицензией eCos.
U-Boot	U-Boot [http://sourceforge.net/projects/u-boot/]	Универсальный загрузчик под лицензией GPL, предоставляет поддержку разных CPU.
ABLE	ABLE bootloader [http://www.simtec.co.uk/products/SWABLE/]	Коммерческий загрузчик со всеобъемлющим набором функций.

3. Обзор

ARM Linux не может быть запущен на машине без небольшого количества машинно-зависимого кода для инициализации системы. ARM Linux ***требует*** код загрузчика сделать очень немногое, хотя несколько загрузчиков предоставляют обширную дополнительную функциональность. Минимальные требования:

- Настроить системную память.

- Загрузить образ ядра по правильному адресу памяти.
- При необходимости загрузить первоначальный RAM диск по правильному адресу в памяти.
- Инициализировать параметры загрузки, чтобы перейти к ядру.
- Получить тип машины ARM Linux.
- Обеспечить вход в ядро с соответствующими значениями регистров.

Как правило, ожидается, что в дополнение к этим основным задачам загрузчик будет инициализировать последовательную или видео консоль для ядра. Действительно, последовательный порт считается почти обязательными в большинстве конфигураций системы.

Каждый из этих шагов будет рассмотрен в последующих разделах.

4. Конфигурирование системной памяти

Ожидается, что загрузчик найдёт и проинициализирует всю память, которую ядро будет использовать для хранения в системе изменяющихся данных. Он выполняет это машинно-зависимым способом. Он может использовать внутренние алгоритмы для автоматического обнаружения и измерения всей оперативной памяти, или может использовать знания о памяти машины, или любой другой метод по усмотрению разработчика загрузчика.

Во всех случаях следует отметить, что все установки выполняются загрузчиком. Ядро не имеет никаких знаний по установке или конфигурации памяти в системе, помимо предусмотренной в загрузчике. Использование machine_fixup() внутри ядра определённо не самое правильное место для этого. Существует чёткое различие в этой области между ответственностью загрузчика и ядром.

Физическое распределение памяти передаётся в ядро с помощью параметра ATAG_MEM. Память не обязательно должны быть полностью непрерывной, хотя и предпочтительно минимальное количество фрагментов. Разрешается иметь несколько блоков ATAG_MEM для нескольких областей памяти. Ядро будет объединять блоки, пришедшие к нему, если они являются смежными физическими областями. Загрузчик может манипулировать памятью с помощью командной строки ядра, используя параметр 'mem=', опции для этого параметра полностью документированы в ***linux/Documentation/kernel-parameters.txt***.

Командная строчка ядра 'mem=' имеет синтаксис mem=<size>[KM][,@<phys_offset>], который позволяет задать размер и расположение физической памяти для определённой области памяти. Это позволяет указать несколько раздельных блоков памяти с разными смещениями, указывая параметр mem= несколько раз.

5. Загрузка образа ядра

Образами ядра, генерируемыми процессом сборки ядра, являются либо файл без сжатия "Image", либо сжатый файл zImage.

Несжатый образ Image, как правило, не используется, так как он не содержит легко идентифицирумое магическое число. Практически повсеместно предпочтение отдаётся использованию сжатого формата zImage.

zImage имеет ряд преимуществ в дополнение к магическому числу. Как правило, декомпрессия образа ***быстрее***, чем чтение с какого-либо внешнего носителя. Целостность образа может быть гарантирована, так как любые ошибки приведут к ошибке декомпрессии.

Ядро знает свою внутреннюю структуру и состояние, которое позволяет получить лучший результат, чем общие внешние методы сжатия.

zImage имеет в своём начале магическое число и некоторую полезную информацию.

Таблица 2. Используемые поля в заголовочном коде zImage

Смещение в zImage	Значение	Описание
0x24	0x016F2818	Магическое число, используемое для идентификации этого zImage Linux для ARM
0x28	начальный адрес	Адрес, где zImage начинается
0x2C	конечный адрес	Адрес, где zImage заканчивается

Смещения начала и конца могут быть использованы для определения длины сжатого изображения (размер = конец - начало). Это используется некоторыми загрузчиками, чтобы определить, были ли добавлены в образ ядра какие-либо данные. Эти данные обычно используются для первоначального RAM диска RAM (initrd). Начальный адрес, как правило, 0, так как код zImage позиционируется независимо.

Код zImage является позиционно-независимым кодом (Position Independent Code, PIC), поэтому может быть загружен куда угодно в пределах имеющегося адресного пространства. Максимальный размер ядра после распаковки 4 мегабайта. Это жёсткое ограничение и будет включать initrd, если была использована цель bootpImage.

Примечание
Хотя zImage может быть расположен в любом месте, следует позаботиться об этом. Старт сжатого ядра требует дополнительной памяти для распаковки образа. Это пространство имеет определённые ограничения.

Код декомпрессии zImage гарантирует, что скомпрессированные данные не будут перезаписаны. Если ядро обнаруживает такой конфликт, оно будет распаковывать образ сразу *после* сжатых данных zImage и перенесёт ядро после декомпрессии. Это, очевидно, имеет последствия, что после области памяти загрузки zImage *должно* быть до 4 мегабайт свободного места после него (максимальный размер несжатого ядра), то есть размещение zImage в том же 4-х мегабайтном банке, как и ZRELADDR, будет, вероятно, не работать, как ожидалось.

Несмотря на возможность разместить zImage в любом месте памяти, традиционно оно загружается по базовому адресу физической памяти плюс смещение 0x8000 (32K). Это оставляет пространство для блока параметров, обычно размещённого со смещением 0x100, нулевой страницы векторов исключений и таблиц страниц. Это соглашение является очень распространённым.

6. Загрузка первоначального RAM диска

Первоначальный RAM диск является общим требованием на многих системах. Он даёт возможность иметь доступной корневую файловую систему без доступа к другим драйверам или конфигурациям. Полную информацию можно получить в ***linux/Documentation/initrd.txt***.

Есть два способа на ARM Linux для получения первоначального RAM диска. Первым из них является создание специального целевого bootpImage, который берёт первоначальный RAM диск во время ***сборки*** и добавляет его в zImage. Этот метод обладает тем преимуществом, что не требует вмешательства загрузчика, но требует знания процессом сборки ядра физического адреса для размещения ramdisk (используя определение INITRD_PHYS). Применяется жёсткое ограничение в 4 мегабайта на размер несжатого ядра и initrd. Из-за этих ограничений этот способ на практике используется редко.

Вторым и гораздо более широко используемым методом является размещение загрузчиком данного исходного образа ramdisk, полученного с любого носителя, в указанное место в памяти. Это место передаётся в ядро с помощью ATAG_INITRD2 и ATAG_RAMDISK.

Традиционно initrd размещается со смещением 8 мегабайт от начала физической памяти. Там, где он находится, должно быть достаточно памяти после загрузки для распаковки начального ramdisk в реальный ramdisk, то есть достаточно памяти для zImage + распакованный zImage + initrd + распакованный ramdisk. Занимаемая сжатым начальным ramdisk память будет освобождена после окончания декомпрессии. Ограничениями к позиции электронного диска являются:

- Она должна лежать полностью в пределах одной области памяти (не должна пересекать регионы, заданные различными параметрами ATAG_MEM)
- Она должна быть согласована с границей страницы (обычно, 4К)
- Она не должна конфликтовать с памятью, используемой заголовочным кодом zImage, используемым для распаковки ядра или она будет перезаписана, как не прошедшая проверки.

Замечание. Дополнительно появился способ делать это с помощью initramfs.

7. Инициализация консоли

Консоль настоятельно рекомендуется, как метод увидеть, какие действия выполняет ядро при инициализации системы. Это может быть любое устройство ввода-вывода с подходящим драйвером, наиболее распространёнными являются случаи драйвера кадрового буфера видео или драйвер последовательного порта. Системы, на которых работает ARM Linux, как правило, почти всегда обеспечивают последовательный порт консоли.

Загрузчик должен проинициализировать и разрешить один последовательный порт на машине. Это включает разрешение управления питанием и тому подобное для использования порта. Это позволяет драйверу последовательного порта ядра автоматически определить, какой последовательный порт он должен использовать для консоли ядра (обычно используемым для отладки или коммуникации с машиной).

В качестве альтернативы, загрузчик может передать ядру соответствующую опцию 'console=' через параметр командной строки с указанием порта, а также параметры управления коммуникацией, как описано в ***linux/Documentation/kernel-parameters.txt***.

8. Параметры ядра

Загрузчик должен передать параметры ядру для описания выполненных установок, размера и формы памяти в системе и, опционально, многие другие величины.

Список тэгов должен отвечать следующим ограничениям:

- Список должен быть сохранён в оперативную память и размещён в области памяти, где ни декомпрессор ядра, ни манипуляции initrd не перезапишут его. Рекомендуется размещение в первых 16 Кб оперативной памяти, обычно, в начале физической памяти плюс 0x100 (который оставляет место нулевой странице векторов исключений). Физический адрес списка тэгов должен быть помещен в R2 при входе в ядро, но исторически это не было обязательным и ядро будет использовать фиксированные значения в начале физической памяти плюс 0x100. На это ***не*** следует полагаться в будущем.
- Список не должен выходить за границы области 0x4000, где создаётся начальная таблица трансляции страниц. Ядро не выполняет никаких проверок границ и перезапишет список параметров, если он сделает это.
- Список должен быть выровнен к границам слова (32 бита, 4 байта) (если не используете рекомендованные размещения).
- Список должен начинаться с ATAG_CORE и заканчиваться ATAG_NONE.
- Список должен содержать по крайней мере один ATAG_MEM

Каждый тэг в списке состоит из заголовка, содержащего две беззнаковых 32-битных величины: размера тэга (32 бита, 4 байтное слово) и значение тэга.

```
struct atag_header {
    u32 size; /* размер тэга в словах, включая этот заголовок */
    u32 tag;  /* значение тэга */
};
```

За каждым заголовочным тэгом следуют данные, связанные с этим тэгом, за исключением ATAG_NONE, который не имеет данных и ATAG_CORE, где данные не является обязательными. Размер данных определяется размером поля в заголовке, минимальный размер равен 2, так как размер заголовка включён в эту величину. ATAG_NONE уникален тем, что его поле размера (size) устанавливается в ноль.

Тэг может содержать дополнительные данные после обязательных структур при условии, что размер (size) скорректирован для покрытия дополнительной информации, это даёт возможность дальнейшего расширения и загрузчик сможет продлить данные, предоставляемые ядру. Например, загрузчик может предоставить дополнительную информацию о серийных номерах в ATAG_SERIAL, которые могут затем быть интерпретированы модифицированным ядром.

Порядок тэгов в списке параметров не имеет значения, они могут появиться столько раз, сколько требуется, хотя толкование дублирующихся тэгов зависит от обстоятельств.

Данные для каждого отдельного тэга описаны в Приложении А, справочный разделе "Значения Тэгов" 10.

Таблица 3. Список используемых тэгов.

Имя тэга	Значение	Размер	Описание
ATAG_NONE	0x00000000	2	Пустой тэг, используемый в конце списка
ATAG_CORE	0x5441000	5 (2 if empty)	Первый тэг, используемый в начале списка

	1		
ATAG_MEM	0x54410002	4	Описывает физическую область памяти
ATAG_VIDEOTEXT	0x54410003	5	Описывает текстовый VGA дисплей
ATAG_RAMDISK	0x54410004	5	Описывает как в ядре будет использоваться ramdisk
ATAG_INITRD2	0x54420005	4	Описывает где размещается в памяти скомпрессированный образ ramdisk
ATAG_SERIAL	0x54410006	4	64 битный серийный номер
ATAG_REVISION	0x54410007	3	32 битный номер ревизии платы
ATAG_VIDEOLFB	0x54410008	8	Начальные значения кадровых буферов типа vesafb-type
ATAG_CMDLINE	0x54410009	2 + ((length_of_cmdline + 3) / 4)	Командная строка для передачи ядру

Структура может быть определена следующим образом:

```
struct atag {
    struct atag_header hdr;
    union {
        struct atag_core core;
        struct atag_mem mem;
        struct atag_videotext videotext;
        struct atag_ramdisk ramdisk;
        struct atag_initrd2 initrd2;
        struct atag_serialnr serialnr;
        struct atag_revision revision;
        struct atag_videolfb videolfb;
        struct atag_cmdline cmdline;
    } u;
};
```

После того, как эти структуры были определены, необходимо их определить примерно так, как показано:

```
#define tag_next(t) ((struct tag *)((u32 *)(t) + (t)->hdr.size))
#define tag_size(type) ((sizeof(struct tag_header) + sizeof(struct type)) >>
2)
static struct atag *params; /* используется как указатель на текущий тэг */

static void
setup_core_tag(void * address,long pagesize)
{
    params = (struct tag *)address;         /* Начальные параметры для старта
*/
    params->hdr.tag = ATAG_CORE;            /* начинаем с основного тэга */
    params->hdr.size = tag_size(atag_core); /* размер тэга */
```

```
    params->u.core.flags = 1;                /* обеспечить только чтение */
    params->u.core.pagesize = pagesize;      /* размер системной страницы (4k)
*/
    params->u.core.rootdev = 0;              /* нулевое корневое устройство
(обычно перезаписывается)
    params = tag_next(params);               /* передвинуть указатель на
следующий тэг */
}

static void
setup_mem_tag(u32_t start, u32_t len)
{
    params->hdr.tag = ATAG_MEM;             /* Тэг памяти */
    params->hdr.size = tag_size(atag_mem); /* размер тэга */
    params->u.mem.start = start;            /* Начало области памяти
(физический адрес)
    params->u.mem.size = len;               /* Размер области */
    params = tag_next(params);              /* передвинуть указатель на
следующий тэг */
}

static void
setup_end_tag(void)
{
    params->hdr.tag = ATAG_NONE; /* Пустой тэг заканчивает список */
    params->hdr.size = 0;        /* нулевая длина */
}

static void
setup_tags(parameters)
{
    setup_core_tag(0x100, 4096);         /* стандартный основной тэг, размер
страницы 4k */
    setup_mem_tag(0x10000000, 0x400000); /* 64Mb с адреса 0x10000000 */
    setup_mem_tag(0x18000000, 0x400000); /* 64Mb с адреса 0x18000000 */
    setup_end_tag(void);                 /* завершающий тэг */
}
```

Этот завершённый фрагмент кода иллюстрирует абсолютные минимальные требования для набора параметров и предназначен для демонстрации концепций, высказанных ранее в этом разделе. Реальный загрузчик, вероятно, установит дополнительное значения и, вероятно, протестирует память в реальной системе, вместо использования фиксированных значений. Более полный пример можно найти в Приложении B, "Полный пример" 16.

9. Получение типа машины Linux

Только одна дополнительная информация необходима загрузчику - предоставить тип машины, это простое число, уникальное для каждой системы ARM, часто называемое MACH_TYPE.

Номер типа машины получается через сайт ARM Linux Machine Registry [http://www.arm.linux.org.uk/developer/machines/]. Тип машины должен быть получен насколько возможно рано в жизни проекта, это имеет ряд последствий для портирования ядром самого себя (машинно-

зависимые определения и т.д.) и изменение определений потом может привести к ряду нежелательных вопросов. Эти значения представлены списком определений в исходных кодах ядра (***linux/arch/arm/tools/mach-types***).

Загрузчик должен каким-то способом получить значение типа машины. Или это жёстко заданное значение или алгоритм, который проверяет подключенное оборудование. Реализация является полностью системо-зависимой и выходит за рамки этого документа.

10. Запуск ядра

После того, как загрузчик выполнил все другие шаги, он должен начать выполнение ядра с правильными значениями в регистрах процессора.

Входными требованиями являются:

- Процессор должен быть в режиме SVC (supervisor) с отключенными прерываниями IRQ и FIQ.
- MMU должен быть ***выключен***, то есть код работает с физической оперативной памятью без трансляции адресов.
- Кэш данных должен быть ***выключен***.
- Кэш инструкций может быть либо включен, либо выключен.
- Регистр 0 процессора должен быть 0.
- Регистр 1 процессора должен содержать тип машины ARM Linux.
- Регистр 2 процессора должен содержать физический адрес списка параметров.

Ожидается, что загрузчик вызовет образ ядра, делая переход прямо на первую инструкцию образа ядра.

А. Значения Тэгов

ATAG_CORE

ATAG_CORE -- Начальный тэг, используемый в начале списка

ATAG_CORE

Значение

0x54410001

Размер

5 (2, если нет данных)

Поля структуры

```
struct atag_core {
    u32 flags;    /* бит 0 = только чтение */
    u32 pagesize; /* страница памяти системы (обычно 4k) */
    u32 rootdev;  /* номер корневого устройства */
};
```

Описание

Этот тэг ***должен*** использоваться, чтобы начать список, в нём содержатся основные сведения, которые должен передать любой загрузчик, тэг длиной 2 указывает, что тэг не имеет подключенной структуры.

ATAG_NONE

ATAG_NONE -- Пустой тэг, используемый в конце списка

ATAG_NONE

Значение

0x00000000

Размер

2

Поля структуры

Нет

Описание

Этот тэг используется для обозначения конца списка. Он уникален тем, что размер поля в заголовке должен быть установлен в 0 (не 2).

ATAG_MEM

ATAG_MEM -- Тэг, используемый для описания физической области памяти.

ATAG_MEM

Значение

0x54410002

Размер

4

Поля структуры

```
struct atag_mem {
    u32 size;   /* размер области */
    u32 start; /* физический адрес старта */
};
```

Описание

Описывает область физической памяти для использования ядром.

ATAG_VIDEOTEXT

ATAG_VIDEOTEXT -- Тэг, используемый для описания VGA дисплеев текстового типа

ATAG_VIDEOTEXT

Значение

0x54410003

Размер

5

Поля структуры

```
struct atag_videotext {
    u8 x; /* ширина дисплея */
    u8 y; /* высота дисплея */
    u16 video_page;
    u8 video_mode;
    u8 video_cols;
    u16 video_ega_bx;
    u8 video_lines;
    u8 video_isvga;
    u16 video_points;
};
```

Описание

ATAG_RAMDISK

ATAG_RAMDISK -- Тэг, описывающий как будет использоваться ядром ramdisk

ATAG_RAMDISK

Значение

0x54410004

Размер

5

Поля структуры

```
struct atag_ramdisk {
    u32 flags; /* бит 0 = загрузить, бит 1 = запросить */
    u32 size;  /* размер декомпрессированного ramdisk в _кило_ байтах */
    u32 start; /* начальный блок образа RAM диск, расположенного на дискете */
};
```

Описание

Описывает, как (начальный) ramdisk будет конфигурироваться ядром, в частности, это позволяет загрузчику быть уверенным, что ramdisk будет достаточно большим, чтобы вместить ***декомпрессированный*** образ начального ramdisk, он обеспечивает это передачей ATAG_INITRD2.

ATAG_INITRD2

ATAG_INITRD2 -- Тэг, описывающий физическое расположение скомпрессированного образа ramdisk

ATAG_INITRD2

Значение

0x54420005

Размер

4

Поля структуры

```
struct atag_initrd2 {
    u32 start; /* физический стартовый адрес */
    u32 size;  /* размер скомпрессированного образа ramdisk в байтах */
};
```

Описание

Местонахождение сжатого образа ramdisk, обычно в сочетании с ATAG_RAMDISK. Может быть использован в качестве начальной корневой файловой системы с добавлением параметра командной строки 'root=/dev/ram'.

Этот тег ***заменяет*** собой оригинальный ATAG_INITRD, который использовал виртуальную адресацию, это была ошибка и на некоторых системах возникали проблемы. Все новые загрузчики должны отдавать предпочтение в использовании этому тэгу.

ATAG_SERIAL

ATAG_SERIAL -- Тэг с 64-х разрядным номером платы

ATAG_SERIAL

Значение

0x54410006

Размер

4

Поля структуры

```
struct atag_serialnr {
    u32 low;
    u32 high;
};
```

Описание

ATAG_REVISION

ATAG_REVISION -- Тэг для версии платы

ATAG_REVISION

Значение

0x54410007

Размер

3

Поля структуры

```
struct atag_revision {
    u32 rev;
};
```

Описание

ATAG_VIDEOLFB

ATAG_VIDEOLFB -- Тэг, описывающий параметры дисплея с кадровым буфером

ATAG_VIDEOLFB

Значение

0x54410008

Размер

8

Поля структуры

```
struct atag_videolfb {
    u16 lfb_width;
    u16 lfb_height;
    u16 lfb_depth;
    u16 lfb_linelength;
    u32 lfb_base;
    u32 lfb_size;
    u8 red_size;
    u8 red_pos;
    u8 green_size;
    u8 green_pos;
    u8 blue_size;
    u8 blue_pos;
    u8 rsvd_size;
```

```
    u8 rsvd_pos;
};
```

Описание

ATAG_CMDLINE

ATAG_CMDLINE -- Тэг, используемый для передачи командной строки ядру

ATAG_CMDLINE

Значение

0x54410009

Размер

2 + ((length_of_cmdline + 3) / 4)

Поля структуры

```
struct atag_cmdline {
    char cmdline[1]; /* это минимальный размер */
};
```

Описание

Используется для передачи параметров командной строки ядру. Командная строка должна заканчиваться NULL. Переменная length_of_cmdline должна включать этот терминатор.

B. Полный пример

Это рабочий пример простого загрузчика и он показывает всю информацию, описанную в этом документе. Для реального загрузчика может потребоваться больше кода, этот пример чисто иллюстративен.

Код этого примера распространяется под лицензией BSD, может быть свободно скопирован и использован, если это необходимо.

```
/* example.c
 * пример кода загрузчика для ARM Linux
 * этот пример распространяется под лицензией BSD
 */

/* список возможных тэгов */
#define ATAG_NONE       0x00000000
#define ATAG_CORE       0x54410001
#define ATAG_MEM        0x54410002
#define ATAG_VIDEOTEXT 0x54410003
#define ATAG_RAMDISK    0x54410004
#define ATAG_INITRD2    0x54420005
#define ATAG_SERIAL     0x54410006
#define ATAG_REVISION   0x54410007
#define ATAG_VIDEOLFB   0x54410008
#define ATAG_CMDLINE    0x54410009

/* структуры для каждого тэга */
struct atag_header {
    u32 size; /* размер тэга в словах, включая этот заголовок */
    u32 tag;  /* тип тэга */
};

struct atag_core {
    u32 flags;
    u32 pagesize;
    u32 rootdev;
};

struct atag_mem {
    u32 size;
    u32 start;
};

struct atag_videotext {
    u8 x;
    u8 y;
    u16 video_page;
    u8 video_mode;
    u8 video_cols;
    u16 video_ega_bx;
    u8 video_lines;
    u8 video_isvga;
    u16 video_points;
};
```

```
struct atag_ramdisk {
    u32 flags;
    u32 size;
    u32 start;
};

struct atag_initrd2 {
    u32 start;
    u32 size;
};

struct atag_serialnr {
    u32 low;
    u32 high;
};

struct atag_revision {
    u32 rev;
};

struct atag_videolfb {
    u16 lfb_width;
    u16 lfb_height;
    u16 lfb_depth;
    u16 lfb_linelength;
    u32 lfb_base;
    u32 lfb_size;
    u8 red_size;
    u8 red_pos;
    u8 green_size;
    u8 green_pos;
    u8 blue_size;
    u8 blue_pos;
    u8 rsvd_size;
    u8 rsvd_pos;
};

struct atag_cmdline {
    char cmdline[1];
};

struct atag {
    struct atag_header hdr;
    union {
        struct atag_core core;
        struct atag_mem mem;
        struct atag_videotext videotext;
        struct atag_ramdisk ramdisk;
        struct atag_initrd2 initrd2;
        struct atag_serialnr serialnr;
        struct atag_revision revision;
        struct atag_videolfb videolfb;
        struct atag_cmdline cmdline;
    } u;
```

```
};

#define tag_next(t) ((struct tag *)((u32 *)(t) + (t)->hdr.size))
#define tag_size(type) ((sizeof(struct tag_header) + sizeof(struct type)) >>
2)
static struct atag *params; /* указывает адрес текущего тэга */

static void
setup_core_tag(void * address,long pagesize)
{
    params = (struct tag *)address; /* Начальные параметры для старта */
    params->hdr.tag = ATAG_CORE; /* начинаем с основного тэга */
    params->hdr.size = tag_size(atag_core); /* размер тэга */
    params->u.core.flags = 1; /* обеспечить только чтение */
    params->u.core.pagesize = pagesize; /* размер системной страницы (4k) */
    params->u.core.rootdev = 0; /* нулевое корневое устройство (обычно
перезаписывается)
    params = tag_next(params); /* передвинуть указатель на следующий тэг */
}

static void
setup_ramdisk_tag(u32_t size)
{
    params->hdr.tag = ATAG_RAMDISK; /* тэг Ramdisk-a */
    params->hdr.size = tag_size(atag_ramdisk); /* размер тэга */
    params->u.ramdisk.flags = 0; /* Загрузить ramdisk */
    params->u.ramdisk.size = size; /* Размер декомпрессированного ramdisk */
    params->u.ramdisk.start = 0; /* Не используется */
    params = tag_next(params); /* передвинуть указатель на следующий тэг */
}

static void
setup_initrd2_tag(u32_t start, u32_t size)
{
    params->hdr.tag = ATAG_INITRD2; /* Тэг Initrd2 */
    params->hdr.size = tag_size(atag_initrd2); /* размер тэга */
    params->u.initrd2.start = start; /* физический старт */
    params->u.initrd2.size = size; /* размер скомпрессированного ramdisk */
    params = tag_next(params); /* передвинуть указатель на следующий тэг */
}

static void
setup_mem_tag(u32_t start, u32_t len)
{
    params->hdr.tag = ATAG_MEM; /* Тэг памяти */
    params->hdr.size = tag_size(atag_mem); /* размер тэга */
    params->u.mem.start = start; /* Начало области памяти (физический адрес)
    params->u.mem.size = len; /* Размер области */
    params = tag_next(params); /* передвинуть указатель на следующий тэг */
}

static void
setup_cmdline_tag(const char * line)
{
    int linelen = strlen(line);
```

```
    if(!linelen)
        return; /* не вставлять тэг для пустой строки */
    params->hdr.tag = ATAG_CMDLINE; /* Тэг командной строки */
    params->hdr.size = (sizeof(struct atag_header) + linelen + 1 + 4) >> 2; /*
размер тэга */
    strcpy(params->u.cmdline.cmdline,line); /* поместить командную строчку в
тэг */
    params = tag_next(params); /* передвинуть указатель на следующий тэг */
}

static void
setup_end_tag(void)
{
    params->hdr.tag = ATAG_NONE; /* Пустой тэг заканчивает список */
    params->hdr.size = 0; /* нулевая длина */
}

#define DRAM_BASE 0x10000000
#define ZIMAGE_LOAD_ADDRESS DRAM_BASE + 0x8000
#define INITRD_LOAD_ADDRESS DRAM_BASE + 0x800000

static void
setup_tags(parameters)
{
    setup_core_tag(parameters, 4096); /* стандартный основной тэг, размер
страницы 4k */
    setup_mem_tag(DRAM_BASE, 0x4000000); /* 64Mb с адреса 0x10000000 */
    setup_mem_tag(DRAM_BASE + 0x8000000, 0x4000000); /* 64Mb с адреса
0x18000000 */
    setup_ramdisk_tag(4096); /* создать 4Mb ramdisk */
    setup_initrd2_tag(INITRD_LOAD_ADDRESS, 0x100000); /* компрессированные
данные 1Mb */
    setup_cmdline_tag("root=/dev/ram0"); /* командная строчка, устанавливающая
корневое устройство */
    setup_end_tag(void); /* завершающий тэг */
}

int
start_linux(char *name,char *rdname)
{
    void (*theKernel)(int zero, int arch, u32 params);
    u32 exec_at = (u32)-1;
    u32 parm_at = (u32)-1;
    u32 machine_type;
    exec_at = ZIMAGE_LOAD_ADDRESS;
    parm_at = DRAM_BASE + 0x100
    load_image(name, exec_at); /* скопировать образ в RAM */
    load_image(rdname, INITRD_LOAD_ADDRESS); /* скопировать образ начального
ramdisk в RAM */
    setup_tags(parm_at); /* установить параметры */
    machine_type = get_mach_type(); /* получить тип машины */
    irq_shutdown(); /* остановить прерывания */
    cpu_op(CPUOP_MMUCHANGE, NULL); /* выключить MMU */
    theKernel = (void (*)(int, int, u32))exec_at; /* установить адрес ядра */
```

```
    theKernel(0, machine_type, parm_at); /* передать управление ядру с
установленными регистрами */
    return 0;
}
```

Библиография

ARM Linux website Documentation [http://www.arm.linux.org.uk/developer/]. Russell M King.

Linux Kernel Documentation/arm/booting.txt [http://www.arm.linux.org.uk/developer/booting.php]. Russell M King.

Setting R2 correctly for booting the kernel [http://lists.arm.linux.org.uk/pipermail/linux-arm-kernel/2003-January/013126.html] (explanation of booting requirements). Russell M King.

Wookey's post summarising booting [http://lists.arm.linux.org.uk/pipermail/linux-arm-kernel/2002-April/008700.html]. Wookey.

Makefile defines and symbols [http://lists.arm.linux.org.uk/pipermail/linux-arm-kernel/2001-July/004064.html]. Russell M King.

Bootloader guide [http://www.aleph1.co.uk/armlinux/docs/ARMbooting/t1.html]. Wookey.

Kernel boot order [http://lists.arm.linux.org.uk/pipermail/linux-arm-kernel/2001-October/005212.html]. Russell M King.

Advice for head.S Debugging [http://lists.arm.linux.org.uk/pipermail/linux-arm-kernel/2002-January/006730.html]. Russell M King.

Linux kernel 2.4 startup [http://billgatliff.com/articles/emb-linux/startup.html/index.html]. Bill Gatliff.

Blob bootloader [http://www.sf.net/projects/blob/]. Erik Mouw.

Blob bootloader on lart [http://www.lart.tudelft.nl/lartware/blob/]. Erik Mouw.