Санкт-Петербургский политехнический университет Петра Великого
Институт Информационных Технологий и Управления
Кафедра компьютерных систем и программных технологий

Отчёт по практической работе
по предмету «Системное программное обеспечение»

Процесс загрузки операционной системы Linux

Работу выполнил студент гр. 53501/3 _____ Мартынов С. А.

Работу принял преподаватель _____ Душутина Е. В.

Санкт-Петербург
2015

Содержание

Введение

Большинство компьютерных систем могут исполнять только команды, находящиеся в оперативной памяти компьютера, в то время как современные операционные системы в большинстве случаев хранятся на жёстких дисках, загрузочных CD-ROM, USB дисках или в локальной сети.

После включения компьютера в его оперативной памяти нет операционной системы. Само по себе, без операционной системы, аппаратное обеспечение компьютера не может выполнять сложные действия, такие как, например, загрузку программы в память. Таким образом мы сталкиваемся с парадоксом: для того, чтобы загрузить операционную систему в память, мы уже должны иметь операционную систему в памяти.

Решением данного парадокса является использование специальной компьютерной программы, называемой начальным загрузчиком. Эта программа не обладает всей функциональностью операционной системы, но её достаточно для того, чтобы загрузить другую программу, которая будет загружать операционную систему. Часто используется многоуровневая загрузка, в которой несколько небольших программ вызывают друг друга до тех пор, пока одна из них не загрузит операционную систему.

Рассмотрим стандартный процесс загрузки Linux системы с момента подачи питания до получения доступа к командному интерпритатору.

1 BIOS

При включении питания компьютера, в его оперативной памяти отсутствуют какие-либо данные, поэтому управление компьютером может осуществляться только аппаратным обеспечением. На Intel-овских платформах начальная загрузка операционной системы осуществляется посредством так называемой "базовой системы ввода/вывода"или BIOS.

После включения компьютера блок питания проверяет все необходимые уровни напряжений. Если все уровни напряжений соответствуют номинальным, то на материнскую плату поступит сигнал PowerGood. До появления этого сигнала на вход процессора подается сигнал RESET, который удерживает процессор в сброшенном состоянии. Но после получения сигнала PowerGood от блока питания сигнал RESET будет снят и процессор начнет выполнять свои первые инструкции. При этом процессор стартует от вполне известного состояния: командный регистр CS содержит 0xFFFF, указатель команд (регистр IP) содержит 0, сегментные регистры данных и стека содержат 0. Таким образом, после снятия RESET процессор в реальном режиме выполняет инструкции, размещающиеся в области ROM BIOS, начинающейся с адреса FFFF:0000 (физический адрес, соответственно, - 0xFFFF0). Размер этой области, очевидно, составляет 16 байт, вплоть до конца максимально адресуемого адресного пространства в реальном режиме - 0xFFFFF. По этому адресу располагается инструкция перехода на реально исполняемый код BIOS.

По соображениям снижения стоимости код BIOS в относительно современных материнских платах хранится в постоянной памяти (ПЗУ) в сжатом виде. Только небольшая его часть, используемая на самых первых этапах загрузки, является непосредственно исполняемой. Поэтому первая задача, которая решается сразу после включения питания, заключается в том, чтобы инициализировать контроллер DRAM, декомпрессировать основной код BIOS и загрузить его в ту область оперативной памяти (RAM), которую именуют "теневой"(shadow RAM).

Эта область затем защищается от записи и управление передается на записанный в нее исполняемый код BIOS. Теневая память в ходе дальнейшей работы отдана в полное владение чипсета материнской платы; операционная система к ней доступа не имеет. Но аппаратными средствами обеспечивается отображение теневой памяти на те области, которые в реальном режиме работы доступны для старых операционных систем типа MS-DOS, так что последние обнаруживают код BIOS именно там, где ожидают его найти.

Исполняемый код BIOS вначале реализует функцию начального самотестирования (POST - Power-On Self Test). При этом тестируются процессор, память и системные средства ввода/вывода, а также производится конфигурирование программно-управляемых аппаратных средств компьютера. Кроме того производится поиск и обнаружение перифе-

рийных устройств. При этом производится сравнение установок, записанных в CMOS (Complementary Metal Oxide Semiconductor) с тем, что реально обнаружено в системе. Некоторые несовпадения, например, различие типов флоппи-дисковода, могут быть допустимы и процесс загрузки продолжится. Другие ошибки, например, отсутствие видеокарты, приводят к невозможности дальнейшей загрузки. Но все сообщения о выявленных на этом этапе ошибках сводятся только к тому, что раздастся несколько коротких звуковых сигналов (каждый вендор имеет свой код звуковых ошибок, который нужно смотреть по документации на материнскую плату).

Некоторые типы периферийных устройств могут содержать расширения BIOS в собственных ПЗУ. В таком случае устанавливаются соответствующие ссылки на эти расширения. Основная причина, по которой это необходимо, заключается в том, что по историческим причинам размер первичного загрузчика (загрузчика первого этапа) на персональных компьютерах ограничен величиной 446 байт. Этого явно недостаточно для того, чтобы включить в этот загрузчик драйверы всех периферийных устройств (например, дисплея и устройств хранения данных), которые могут понадобиться на этапе начальной загрузки системы. Тем более, что драйверы могут различаться для однотипных устройств от разных производителей. Поэтому функция загрузки некоторых драйверов возлагается на BIOS.

После завершения процедуры самотестирования та часть кода BIOS, которая реализует процедуры самотестирования (POST), удаляется из оперативной памяти. Оставшаяся часть BIOS, реализующая функции BIOS (runtime services), остается в ОП. Она будет доступна в последующем для загруженной операционой системы.

Первой задачей той части BIOS, которая осталась в ОП, является поиск активного загрузочного устройства. Список устройств, которые могут являться загрузочными, хранится в энергонезависимой памяти компьютера (CMOS), а порядок просмотра этого списка является одним из настраиваемых параметров BIOS. Загрузочным устройством может быть дискета, CD-ROM, раздел жесткого диска, сетевое устройство или даже USB-устройство (флеш-диск). Поиск загрузочного устройства осуществляется путем вызова прерывания INT19h BIOS.

Процедура обработки прерывания INT19h состоит в том, что считывается сектор с координатами Cylinder:0 Head:0 Sector:1 на очередном устройстве, его содержимое помещается в ОП по адресу 0000:7C00h, после чего осуществляется проверка, является ли этот сектор загрузочным, то есть содержит ли он код первичного загрузчика. Загрузочные сектора помечаются "волшебным"числом 0x55AA в позиции 0x1FE = 510. Это последние два байта сектора. Наличие (или отсутствие) такого кода в последних байтах сектора позволяет программе BIOS решить, является ли данное устройство загрузочным.

Как только программа обработки прерывания обнаружит, что загруженный в память сектор содержит это самое "магическое число"0x55AA, управление передается на начало этого сектора (по абсолютному адресу 0000:7C00h). Дальнейшие события зависят от того, где обнаружен загрузочный сектор - на жестком диске или на одном из других устройств: дискете, CD или flash-диске (на всех этих устройствах обычно создается образ загрузочной дискеты).

Если при проверке загрузочный сектор не обнаружен ни на одном устройстве, вызывается прерывание INT18h. Когда-то (в первых персональных компьютерах, производимых компанией IBM) это прерывание служило для вызова интерпретатора ROM-BASIC, который дальше управлял работой компьютера. Клоны IBM-PC не имеют BASIC в ROM-памяти и теперь это прерывание используют для организации загрузки по сети. Но мы не будем рассматривать эту ветку развития событий и вернемся к тому случаю, когда прерывание INT19h обнаружило загрузочный диск и передало управление находящейся в нем программе.

Примечание: Как следует из приведенного выше описания, BIOS выполняет массу работы по тестированию системы. Ядро Linux потом повторно проделывает всю эту работу. Как правило, после загрузки ядра большинство функций BIOS не используется (хотя есть некоторые исключения) и, тем не менее, этот уже бесполезный код BIOS сохраняется в "теневой"памяти компьютера.

Смещение	Размер	Содержание
0x000	446 байт	Главная загрузочная запись (Master Boot Record)
0x1BE	64 байта	Таблица разбиения диска
0x1FE	2 байта	"Магическое число"(0x55AA)

Таблица 1: Структура главного загрузочного сектора.

2 Master Boot Record и Boot Record

После того, как BIOS нашел загрузочное устройство, он передаёт управление программе, которая находилась в самом первом секторе этого диска или дискеты (физический адрес: цилиндр 0, головка 0, сектор 1). Теперь эта программа загружена в память и именно она управляет ходом дальнейшей загрузки. Со времен MS-DOS эту программу принято называть загрузочной записью (Boot Record), а первый сектор любого диска или дискеты - загрузочным сектором (Boot Sector).

Размер сектора на устройствах хранения данных равен 512 байтам. Существует три варианта загрузки ядра Linux[Т.Айвазяна]:

- с помощью загрузочного сектора Linux, загружающего непосредственно ядро (этот загрузочный сектор);
- с помощью специального загрузчика типа LILO или GRUB;
- с помощью программ, загружающих Linux из другой ОС.

Рассматривать только второй вариант, как наиболее распространенный.

С появлением жестких дисков большого объема, которые стали разбивать на разделы, небольшой загрузчик, размещаемый в загрузочном секторе и загружающий непосредственно ядро, перестал справляться с возросшим объемом задач. Надо было не просто загрузить файл с определенного физического адреса, но и найти загрузочный раздел перед этим. К тому же часть места, отведенного для загрузочной записи, отняла таблица разделов жесткого диска. Поэтому старая загрузочная запись была перенесена в первый сектор так называемого "активного"раздела, а в самый первый сектор на жестком диске стали записывать другую программу, задачей которой было найти "активный"раздел и загрузить программу из этого раздела. Первый сектор жесткого диска стали называть главным загрузочным сектором (а соответствующую программу - главной загрузочной записью или Master Boot Record, MBR). На жёстком диске MBR находится по тому же физическому адресу, что и BOOT-сектор на дискете (цилиндр 0, сторона 0, сектор 1). Его структура представлена в табл. 1

Это "Магическое число"0x55AA, является признаком того, что диск является загрузочным. Содержащаяся в MBR такого диска таблица разбиения определяет 4 первичных раздела жесткого диска. Первые 446 байт MBR содержат небольшую программу, а также текст сообщений об ошибках, которые могут возникнуть в ходе ее выполнения.

Основная задача главной загрузочной записи состоит в том, чтобы найти и загрузить в оперативную память собственно загрузчик операционной системы. MBR сканирует таблицу разделов (partition table) в поисках первого (обычно он и единственный) активного раздела (раздела, помеченного как "загрузочный"). Если в таблице разделов активный раздел не обнаружен или хотя бы один раздел содержит неправильную метку, а также если несколько разделов помечены как активные, выдаётся соответствующее сообщение об ошибке. Когда активный раздел найден, программа считывает в оперативную память первый сектор активного раздела.

Практически все загрузчики современных операционных систем состоят из двух частей: загрузчика первого этапа (или первичного загрузчика), который имеет достаточно малый размер, чтобы разместиться в загрузочном секторе, и значительно большего по объему загрузчика 2-го этапа (или вторичного загрузчика), который может храниться уже где угодно на загрузочном носителе, обычно в разделе, содержащем корневую файловую систему. Загрузчик первого этапа может быть размещен как в главном загрузочном секторе диска, так и в загрузочном секторе активного раздела. Если вам приходилось устанавливать Linux, вы знаете, что программа инсталляции предоставляет пользователю такой выбор. Будем пока для определенности считать, что загрузчик 1-го этапа мы поместили в первый сектор активного раздела.

Итак, главная загрузочная запись отыскала активный раздел и загрузила из него в память загрузчик первого этапа, который теперь отвечает за продолжение процесса загрузки. Загрузчик первого этапа имеет такой же небольшой размер, как и код загрузочной записи диска, то есть не более 446 байт. Поэтому и сделать он может не больше, а именно - только загрузить основной загрузчик. Еще более затрудняет ситуацию то, код загрузчика первого этапа пока не имеет доступа к файловой системе и, следовательно, определяет расположение программ на диске, используя только информацию о физических секторах и низкоуровневые вызовы BIOS. Поэтому эта программа имеет единственной целью – загрузку и запуск на выполнение загрузчика второго этапа.

3 Загрузчик 2 этапа операционной системы

Итак, отработали маленькие программы, которые расположены в главной загрузочной записи диска (MBR) и в первом секторе активного раздела (Boot Record), то есть первичный загрузчик ОС. Код первичного загрузчика уже различен в разных операционных системах и является, частью загрузчика этой ОС (или какого-то универсального загрузчика, типа GRUB). Два основных загрузчика – это LILO и GRUB. LILO (LInux LOader) – это "родной"загрузчик операционной системы Linux, который, вытесняется универсальным загрузчиком GRUB (GRand Unified Bootloader).

Главная задача, которая стоит перед загрузчиком любой операционной системы, заключается в том, чтобы перенести в память ядро операционной системы и передать этому ядру управление дальнейшим функционированием компьютера. При этом современные загрузчики позволяют выбрать ядро из нескольких возможных вариантов как одной и той же операционной системы, так и загрузить ядра разных ОС. Некоторые загрузчики (в частности, GRUB) предоставляют пользователю возможность выполнения некоторых команд, то есть они представляют собой уже некоторое подобие командного процессора.

Одна из основных задач ядра – управление железом, которое составляет аппаратную часть компьютера. Поэтому ядро после запуска производит опрос железа. Если драйвер найденного устройства вкомпилирован в ядро, устройство инициализируется. Однако стандартное ядро дистрибутива не может содержать драйверы для всех устройств, имеющихся в компьютере. Драйверы устройств, отсутствующие в ядре, надо загрузить из соответствующих файлов. После загрузки ядро должно запустить процесс init, который. Код программы init лежит где-то на носителе, доступ к которому тоже предоставляет файловая система.

Но файловая система не является частью ядра, драйвер файловой системы надо загрузить с того же диска. Корневая файловая система на загрузочном устройстве может быть самого разного типа (fat, ext2, reiserfs и так далее), может быть сжата, зашифрована. Получается замкнутый круг: чтобы подключить загрузочное устройство, надо получить доступ к файловой систем, а для этого надо подключить загрузочное устройство.

Решение этой проблемы было найдено за счет временного подключения виртуальной файловой системы, образ которой хранится в файле рядом с образом ядра. При этом минимальное число самых необходимых драйверов включается в само ядро, а все остальные размещаются на виртуальном диске, в виде подгружаемых модулей. Когда осуществляется загрузка (в частности, на неизвестном оборудовании) ядро опрашивает аппаратуру и загружает только те модули, которые соответствуют обнаруженной конфигурации. В частности, и LILO и GRUB создают в оперативной памяти виртуальный диск и разворачивают на нем временную корневую файловую систему, содержащую необходимые ядру

драйверы устройств и служебные файлы. Сжатый образ этой файловой системы обычно располагается в одном каталоге с образом ядра в виде файла, содержащего дополнительные компоненты, например, драйверы некоторых устройств.

4 Загрузка и инициализация ядра

Во время загрузки Linux на консоль выводится большое количество сообщений, в том числе и сообщений о действиях, выполняемых ядром при его инициализации. Эти сообщения обычно быстро проскакивают, и их невозможно прочитать. Ещё эти сообщения могут быть скрыты за заставкой. Но все эти сообщения сохраняются и их можно увидеть после завершения загрузки, воспользовавшись командой dmesg. Протоколирование этих сообщений осуществляется демоном протоколирования klogd (он запускается как один из потоков ядра). Klogd сохраняет эти сообщения в специальном буфере ядра, содержимое которого выдается по команде dmesg. Кроме того, klogd передает эти сообщения демону системного протоколирования syslogd, который записывает их в файл /var/log/messages. Благодаря этому они и доступны для просмотра и анализа после завершения запуска системы.

Ядро хранится на диске в сжатом виде. Только его первая часть не сжата. Когда загрузчик перенесет ядро в память, эта несжатая часть ядра выполняет декомпрессию оставшейся части. Несжатая часть кода, которая содержится в начале ядра, написана на ассемблере, ее можно найти в файлах arch/i386/boot/setup.S, arch/i386/boot/video.S и arch/i386/kernel/head.S.

Код, хранящийся в файлах arch/i386/boot/setup.S и arch/i386/boot/video.S, отвечает за получение системной информации от BIOS и размещение ее в подходящих местах системной памяти. Кратко перечислим основные действия, которые выполняет эта часть ядра:

- Проверка того, что ядро загружено правильно. Осуществляется проверкой сигнатуры в конце кода запуска. Если она не найдена, то копируются секторы с запускающим кодом и сигнатура ищется снова. Если опять безуспешно, то выдается сообщение "No setup signature found ...".
- Проверка на возможность работы с верхней памятью. Если образ ядра слишком большой (и поэтому загружен в верхнюю память), а код несжатой части ядра не может работать с образами, расположенными в верхней памяти, то загрузка прекращается и выдачей сообщения "Wrong loader, giving up...".
- Выяснение объема памяти системы. Для этого используется вызов трех различных функций BIOS: E820h, которая позволяет собрать карту памяти, затем E801h, которая вернет 32-битный размер памяти и последней вызывается 88h, которая вернет размер в диапазоне 0-64 МБ. Полученные значения сохраняются для дальнейшего использования.
- Настройка режима видеоадаптера. Возможность выбора видеорежима зависит от

конфигурации ядра. Можно указать специфичный (предопределенный) режим, который будет использован в процессе загрузки ядра, либо запросить меню со списком режимов, из которого пользователь выберет режим по своему желанию.

- Осуществляется подготовка к переключению в защищенный режим: ядро перемещается в подходящее место (если это необходимо), подправляются адреса и значения регистров.
- Производится переключение процессора в защищенный режим.

5 Процесс init и файл /etc/inittab

Если в параметре начальной загрузки "init="не задан запуск какой-то другой программы, после монтирования корневой файловой системы в режиме "только для чтения"ядро запускает процесс init, который является родоначальником всех других процессов в Linux. Сам по себе init ничем не отличается от других программ в системе Linux, просто это первая (и единственная) программа, которая запускается непосредственно ядром, все остальные процессы являются потомками процесса init. Файл программы init вы можете найти в каталоге /sbin среди других исполняемых файлов.

Init отвечает за продолжение процедуры загрузки, и перевод системы от начального состояния, возникающего после загрузки ядра, в стандартное состояние обработки запросов многих пользователей. Основная задача, которая стоит перед init, заключается в том, чтобы запускать в определенной последовательности другие программы в процессе загрузки системы и останавливать процессы в случае переключения уровня выполнения (в частности, при остановке системы). Init выполняет еще массу различных операций, необходимых для дальнейшей работы системы: проверку и монтирование файловых систем, запуск различных служб (демонов), запуск процедур логирования, оболочек пользователей на различных терминалах и т.д.

Производители различных дистрибутивов имеют свои подходы к стилям загрузки и так называемым "уровням выполнения". Уровни выполнения (run levels) – это просто несколько стандартных вариантов загрузки системы, каждый из которых определяет перечень действий, выполняемых процессом init, и состояние системы после загрузки, т. е. конфигурацию запущенных процессов. Уровень выполнения идентифицируется одним символом. В большинстве дистрибутивов ОС Linux используется 6 основных уровней выполнения.

В случае Red Hat, распределение уровней будет следующим:

0 остановка системы

1 однопользовательский режим

2 многопользовательский режим без NFS (то же, что и 3, если компьютер не работает с сетью)

3 полный многопользовательский режим

4 использование не регламентировано

5 запуск системы в графическом режиме

6 перезагрузка системы;

В некоторых дистрибутивах (например, в Debian), кроме того, используется дополнительный уровень S (или s) – примерно то же, что и однопользовательский режим, но S и s используются в основном в скриптах.

Как видите, уровни 0, 1 и 6 зарезервированы для особых случаев. Относительно того, как использовать уровни со 2 по 5, единого мнения не существует. Некоторые системные администраторы используют разные уровни для того, чтобы задать разные варианты работы, например, на одном уровне запускается графический режим, на другом работают в сети и т. д.

Уровень выполнения может быть задан как одна из опций, передаваемых ядру загрузчиком. Обычно единственной причиной, по которой уровень загрузки может быть задан как аргумент при загрузке, является необходимость запуска системы в однопользовательском режиме (уровень выполнения 1) для выполнения каких-то административных задач или в случае повреждения диска. Но если уровень выполнения не задан как опция загрузки, то init будет загружать систему на уровень, заданный в файле /etc/inittab.

Конфигурационный файл /etc/inittab состоит из отдельных строк. Если строка начинается со знака # или пуста, то она игнорируется. Все остальные строки состоят из 4 полей, разделенных двоеточиями:

id:runlevels:action:process

где:

- id – идентификатор строки. Это произвольная комбинация, содержащая от 1 до 4 символов. В файле inittab не может быть двух строк с одинаковыми идентификаторами;
- runlevels – уровни выполнения, на которых эта строка будет задействована. Уровни задаются цифрами или буквами без разделителей, например, 345;
- process – процесс, который должен запускаться на указанных уровнях. Другими словами в этом поле указывается имя программы, вызываемой при переходе на указанные уровни выполнения;
- action – действие.

В поле action стоит ключевое слово, которое определяет дополнительные условия выполнения команды, заданной полем process. Допустимые значения поля action:

- respawn – перезапустить процесс в случае завершения его работы;
- once – выполнить процесс только один раз при переходе на указанный уровень;

- wait – процесс будет запущен один раз при переходе на указанный уровень и init будет ожидать завершения работы этого процесса, прежде, чем продолжать работу;
- sysinit – это ключевое слово обозначает самые первые действия, выполняемые процессом init еще до перехода на какой-либо уровень выполнения (поле id игнорируется). Процессы, помеченные этим словом, запускаются до процессов, помеченных словами boot и bootwait;
- boot – процесс будет запущен на этапе загрузки системы независимо от уровня выполнения;
- bootwait – процесс будет запущен на этапе загрузки системы независимо от уровня выполнения, и init будет дожидаться его завершения;
- initdefault – строка, в которой это слово стоит в поле action, определяет уровень выполнения, на который система переходит по умолчанию. Поле process в этой строке игнорируется. Если уровень выполнения, используемый по умолчанию, не задан, то процесс init будет ждать, пока пользователь, запускающий систему, не введет его с консоли;
- off – игнорировать данный элемент;
- powerwait – позволяет процессу init остановить систему, когда пропало питание. Использование этого слова предполагает, что имеется источник бесперебойного питания (UPS) и программное обеспечение, которое отслеживает состояние UPS и информирует init о том, что питание отключилось;
- ctrlaltdel — разрешает init перезагрузить систему, когда пользователь нажимает комбинацию клавиш <Ctrl>+<Alt>+<Del> на клавиатуре.

Этот список не является исчерпывающим.

6 Инициализационный скрипт rc.sysinit

В обычной ситуации одним из первых действий, выполняемых процессом init является запуск на выполнение скрипта rc.sysinit из каталога /etc/rc.d.

Очень кратко, перечень действий, выполняемых скриптом rc.sysinit, выглядит примерно так (на примере дистрибутива Mandriva Linux):

1. Задаются начальные значения общесистемных переменных PATH, HOSTNAME, HOSTTYPE и т.д., а также значения нескольких переменных, которые будут использоваться на дальнейших этапах загрузки.
2. Определяется, используя содержимое файла /etc/sysconfig/network, должна ли данная система подключаться к какой-то локальной сети. В этом файле просто задается значение переменной NETWORKING (да или нет).
3. Аналогичным образом (то есть путем считывания установок из нескольких файлов, размещаемых в каталоге /etc/sysconfig, и задания каких-то значений переменных) задается необходимость использования usb-устройств, уровень безопасности и т.д.
4. Считываются из файла /etc/init.d/functions определения функций, которые будут использоваться скриптами из каталога /etc/init.d на следующих этапах загрузки (а, возможно, и в работающей системе).
5. Монтируются файловые системы /proc (файловая система, используемая в Linux для определения состояния различных процессов) и /sys.
6. Утилита udev вызывается для создания файлов блоковых устройств в каталоге /dev.
7. Задается системный шрифт.
8. Содержимое командной строки загрузки ядра записывается в переменную cmdline.
9. Выдается сообщение с ссобщением о загружаемой систем и информацией о возможности продолжения загрузки в интерактивном режиме.
10. Перенастраиваются параметры ядра путем запуска команды sysctl. Эта операция выполняется в скрипте rc.sysinit неоднократно по мере изменения каких-то параметров.
11. Восстанавливается системное время и другие временные установки (установку часового пояса и т.д.), ориентируясь на показания датчика времени в BIOS и значения параметров, заданные при инсталляции системы.
12. Загружаются модули, заданные пользователем в /etc/sysconfig/modules/*.modules.

13. Задается сетевое имя компьютера (host name), используемое в процедурах идентификации, таких как NIS (Network Information Service), NIS+ (улучшенная версия NIS) и так далее.

14. Задаются настройки, необходимые для подключения устройств Firewire, USB, RAID-массивов, менеджера логических томов и шифрования диска. Например, проверяется существует ли /proc/bus/usb и является ли он каталогом. Если этот каталог существует, но не указан в файле /proc/mounts, то монтируется файловая система типа usbfs.

15. Проверяется корневая файловая система и, в случае отсутствия проблем, она монтируется в режиме чтения-записи.

16. Задействуется виртуальная память, активизируется и монтируется swap-раздел, указанный в файле /etc/fstab.

17. Проверяются другие файловые системы, перечисленные в /etc/fstab.

18. Монтируются все файловые системы, перечисленные в файле /etc/fstab.

19. Проверяются квоты на использование дискового пространства.

20. Идентифицируется и распознается установленное оборудование, конфигурируются устройства Plug'n'Play, активизируются другие устройства, например, звуковая плата.

21. Проверяет состояние специальных дисковых устройств, таких как RAID (Redundant Array of Inexpensive Disks).

22. Загружаются раскладки клавиатуры. Задается клавиша переключения раскладок.

23. Инициализируется генератор (псевдо)случайных чисел.

24. Запускается оптимизация жесткого диска (вызовом команды hdparm).

25. Обновляется ссылка на файл /boot/System.map (чтобы этот файл соответствовал загружаемому ядру).

26. Содержимое кольцевого буфера ядра выгружается в файл /var/log/dmesg.

27. Производится удаление всех временных файлов, каталогов и переменных, созданных для целей загрузки.

28. Если в командной строке не задан выбор графической оболочки, то для нее задается установка по умолчанию.

7 Скрипт rc и запуск системных сервисов

Скрипт rc.sysinit выполняет те задачи по начальной настройке системы, которые не зависят от уровня выполнения, а скрипт /etc/rc.d/rc, который запускается следующим, должен уже произвести перевод системы на тот уровень выполнения, который задан в файле inittab (или в командной строке). В файле inittab присутствует отдельная строка для каждого уровня выполнения, и в этих строках вызывается один и тот же скрипт, и строки отличаются только аргументом вызова этого скрипта. Этот аргумент (или параметр) и задает уровень выполнения. Но, прежде чем рассматривать функции, выполняемые скриптом rc, надо сказать несколько слов о каталоге /etc/rc.d. Этот каталог играет важную роль в процессе загрузки, поскольку он содержит основные скрипты (программы на языке командного процессора shell), служащие для организации процесса загрузки.

Каталог rc.d содержит следующий набор подкаталогов:

- rc0.d
- rc1.d
- rc2.d
- rc3.d
- rc4.d
- rc5.d
- rc6.d
- init.d

Если просмотреть содержимое подкаталогов rcZ.d, то можно увидить, что в этих подкаталогах содержатся не файлы, а только ссылки на файлы скриптов, находящиеся в других каталогах, а именно (за редким исключением), в каталоге /etc/rc.d/init.d. Подкаталог init.d содержит уже не ссылки, а скрипты, управляющие работой для тех служб, которые обычно запускаются в системе (NFS, sendmail, cron, syslog, httpd и т. п.). Различные скрипты из каталога init.d воспринимают различное число опций (или параметров запуска), но все они понимают опции stop, start и restart.

Если необходимо остановить или наоборот запустить какую-то из стандартных системных служб, можно сделать это, вызвав соответствующий скрипт, например,

```
[root]# /etc/rc.d/rc5.d/S10network start
```

или

```
[root]# /etc/rc.d/init.d/network start
```

В некоторых дистрибутивах существует удобная утилита service, которая позволяет сделать то же самое, не набирая в командной строке полный путь в соответствующему скрипту:

```
[root]# service network start
```

Однако, очевидно, что эта утилита будет корректно работать только в том случае, если скрипт вызова службы находится в стандартном каталоге.

8 Запуск процессов getty и login

Когда выполнение rc N завершится, init будет выполнять команды, заданные в строках с ключевым словом respawn для заданного уровня выполнения. Программы, запущенные с условием respawn будут повторно запускаться после завершения.

Например, в Red Hat для пятого уровня выполнения в файле /etc/inittab обычно прописаны следующие строки:

```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
x:5:respawn:/etc/X11/prefdm -nodaemon
```

Первые 6 из этих строк обеспечивают запуск шести виртуальных консолей. Для этого init порождает процессы, именуемые getty-процессами (от "get tty"— получить терминал), и следит за тем, какой из процессов открывает какой терминал. Каждый getty-процесс устанавливает свою группу процессов, используя вызов системной функции setpgrp, открывает отдельную терминальную линию и обычно приостанавливается во время выполнения функции open до тех пор, пока машина не получит аппаратную связь с терминалом.

Когда функция open возвращает управление, getty-процесс (демон getty) отображает на экране содержимое файла etc/issue и запускает программу login (регистрации в системе), которая требует от пользователей, чтобы они идентифицировали себя указанием регистрационного имени и пароля.

Если пользователь не смог успешно зарегистрироваться, программа регистрации через определенный промежуток времени завершается, закрывая открытую терминальную линию, а процесс init порождает для этой линии следующий getty-процесс, открывающий терминал, вместо прекратившего существование (помните - ключевое слово respawn).

Последняя строка в приведенном выше примере, обозначенная идентификатором x, задает регистрацию пользователя в графическом режиме. Эта строка работает только на 5-ом уровне выполнения, поскольку другие уровни в Red Hat не предполагают использования графики.

9 Старт оболочки Bash

Демон getty запустил программу login, которая выводит предложение ввести имя пользователя. Получив это имя, login обращается к файлу /etc/passwd за получением необходимых данных об этом пользователе, в частности, о его идентификаторе, идентификаторе группы, домашнем каталоге и о том, какую оболочку для него запускать. Одновременно выводится запрос на ввод пароля пользователя.

Когда пользователь введет пароль, программа login считывает его, криптует и сравнивает результат с тем, что лежит в соответствующей строке файла etc/shadow.

Если пользователь ввел правильный пароль, программа login, запускает командный процессор (оболочку). Какую именно оболочку запускать (тут возможен довольно широкий выбор), определяется соответствующим полем в файле /etc/passwd.

Наиболее часто применяемым вариантом командной оболочки является bash.

10 Демонстрация работы загрузчика

Основываясь на предыдущем описании, сделаем не большой загрузчик, который должет поместиться в загрузочную область и выдать информацию о себе. В самом простом виде, он будет выглядить следующим образом.

Листинг 1: Исходный код простого загрузчика

```
/*
 * mbr.c
 *
 *  Created on: Mar 29, 2015
 *      Author: sam
 */

/* XXX these must be at top */
#include "code16gcc.h"
__asm__ ("jmpl  $0, $main\n");

#define __NOINLINE   __attribute__((noinline))
#define __REGPARM    __attribute__ ((regparm(3)))
#define __NORETURN   __attribute__((noreturn))

/* BIOS interrupts must be done with inline assembly */
void __NOINLINE __REGPARM print(const char *s) {
  while (*s) {
    __asm__ __volatile__ ("int  $0x10" : : "a"(0x0E00 | *s), "b"(7))
        ;
    s++;
  }
}
/* and for everything else you can use C! Be it traversing the
   filesystem, or verifying the kernel image etc.*/

void __NORETURN main() {
  print("Hello, World!\r\n:)");
  while (1)
    ;
}
```

Но в таком виде код работать не будет, так как не учитываются следующие проблемы:

- реальный режим работы процессора
- elf файл

Решение в использовании специального шаблона для линкера

Листинг 2: Шаблон для линкера

```
ENTRY(main);
SECTIONS
{
    . = 0x7C00;
    .text : AT(0x7C00)
    {
        _text = .;
        *(.text);
        _text_end = .;
    }
    .data :
    {
        _data = .;
        *(.bss);
        *(.bss*);
        *(.data);
        *(.rodata*);
        *(COMMON)
        _data_end = .;
    }
    .sig : AT(0x7DFE)
    {
        SHORT(0xaa55);
    }
    /DISCARD/ :
    {
        *(.note*);
        *(.iplt*);
        *(.igot*);
```

```
30          *(.rel*);
31          *(.comment);
32  /* add any unwanted sections spewed out by your version of gcc and
       flags here */
33      }
34  }
```

Собрасть получившийся код можно следующим образом

```
$ gcc -c -g -Os -m32 -march=i686 -ffreestanding -Wall -Werror -I. -o mbr.o mbr.c
$ ld -static -melf\_i386 -Tlinker.ld -nostdlib --nmagic -o mbr.elf mbr.o
$ objcopy -O binary mbr.elf mbr.bin

$ dd if=/dev/zero of=floppy.img bs=1024 count=1440
$ dd if=mbr.bin of=floppy.img bs=1 count=512 conv=notrunc

$ qemu-system-i386 -fda floppy.img -boot a
```

Результат загрузки представлен на рисунке 1.

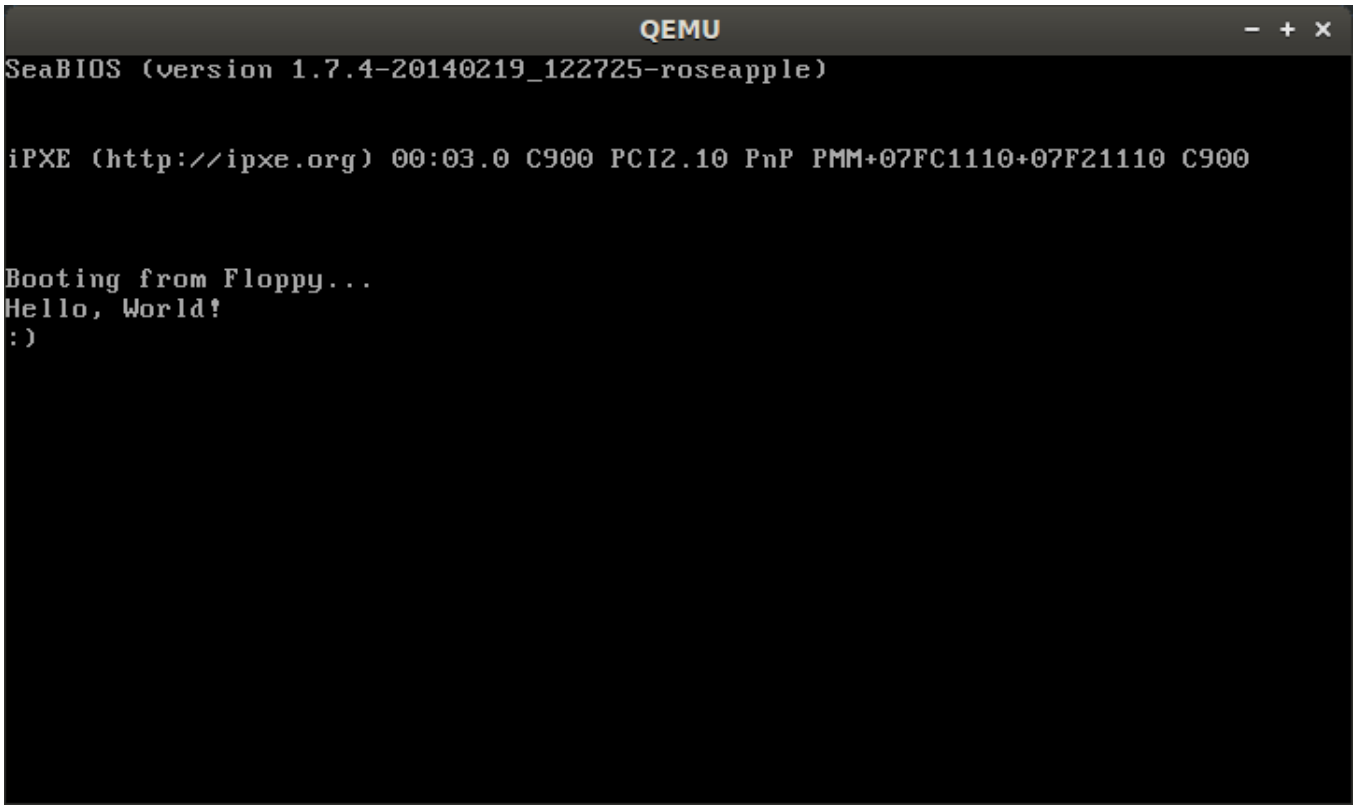

Рис. 1: Запуск загрузчика

Заключение

Рассмотренный в данной работе можно назвать устаревшим, т.к. в современных системах процесс загрузки отличается. Вместо BIOS используется UEFI, а запуск процессов осуществляет systemd.

Однако для многих встраиваемых систем процесс загрузки остался прежним, и это определяется необходимостью экономно расходовать ресурсы.

Список литературы

[1] В.А.Костромин, "Linux для пользователя изд.БХВ-Петербург, 2002 г., серия "Самоучитель".

[2] Немет Э., Снайдер Г., Хейн Т., "Руководство администратора Linux"Изд. Вильямс, 2003 г.

[3] Д.Тейнсли, "Linux и UNIX: программирование в SHELL. Руководство разработчика."Изд.БХВ, Киев, 2001.

[4] А.Робачевский, "Операционная система UNIX". Изд. БХВ-Петербург, СПб, 2002 г.

[5] А.Микляев, "Все настройки BIOS SETUP: Подробное описание всех опций, рекомендации по установке и оптимизации параметров. "РадиоСофт 2004, 192 стр.