# Последовательность загрузки osFree (Предварительный вариант, версия 2)

Замечание: Данный документ находится на стадии разработки. Вы можете присылать замечания и исправления.

Замечание: Спецификация Multiboot поддерживается только в ограниченных пределах. Мы считаем, что не все возможности необходимо поддерживать. Настройка видеорежимов не должна быть задачей загрузчика. Загрузчик должен просто загружать систему и не более.

Замечание: Данный документ использует части документа Multiboot Specification версии 0.6.93

(C) Copyright 2004-2007 osFree project

Данный документ составили: Юрий Прокушев, Валерий Седлецкий, Саша Шмидт.

## Введение

Последовательность загрузки osFree не использует классические решения наподобе GRUB или LILO. GRUB хорошая программа, но имеет ряд недостатков. Главный из них - загрузчик должен знать о структуре файловой системы (ФС). Это означает, что для того, чтобы иметь поддержку новой ФС, вам необходимо, как пользователю, обновить GRUB на более новую версию с поддержкой новой ФС (если такая поддержка существует). А как разработчику, вам потребуется добавить драйвер ФС в ядро и загрузчик. В большинстве случаев это означает различные архитектуры, стиль программирования и среды разработки. Мы не хотим обновлять всю систему для обеспечения поддержки только маленькой функциональности (в сравнении со всей операционной системой). Мы не хотим постоянно переустанавливать или обновлять кучу системных компонент для “указателя мыши с тенью”. Мы хотим, чтобы общая стоимость владения оставалась на минимальном уровне. В результате мы заимствовали идею устанавливаемой файловой системы (УФС) из OS/2. Мы не описываем здесь устройство MicroFSD и MiniFSD, т.к. это задача документа УФС, но не устройства и интерфейсов загрузчика ядра.

## Основополагающая информация (Информативная)

В этом разделе мы попробуем обсудить некоторые вопросы относительно загрузки osFree с диска и процесса системной инициализации. osFree является клоном OS/2, поэтому она должна выполнять задачи теми же способами, что и OS/2. Также мы должны заимствовать хорошие идеи из OS/2 Warp Connect PowerPC Edition (также известной как Workplace OS), так как это был первый пример применения микроядра в OS/2, в котором имелись необходимые особенности для данной системы.

## Последовательность загрузки

В конце процедуры POST ROM BIOS проверяет имеющиеся устройства и передает управление

процедуре прерывания int 19h, которая загружает 1-ый сектор 1-ого загрузочного устройства (дискета, винчестер или др.). Если загрузка производится с винчестера, то главная загрузочная запись (MBR - Master Boot Record) загружается из 1-ого сектора. ROM BIOS загружает его в адрес 0x07c0:0x0000. MBR содержит вход во внесистемный загрузчик (NSB - NonSystem Bootstrap) и таблицу разделов (PT - Partition Table). Код NSB перемещает MBR в 0x0060:0x0000 и загружает загрузочный сектор винчестера в те же адреса (0x07c0:0x0000), куда была загружена MBR. Раздел начальной загрузки находится в таблице разделов, которая содержится в конце MBR. Таблица разделов содержит четыре дескриптора раздела, каждый из которых содержит флаг активности в 1-ом байте структуры дескриптора. Если этот байт равен 80h, то раздел активен, если он соответствует 00h, то выделенные раздел не активен. Затем MBR передает управление загрузочному сектору, который загружает операционную систему. Эта часть загрузки такая же как и в других операционных системах PC, включая Windows, OS/2 и Linux, когда процесс загрузки происходит из активного основного раздела.

Для того, чтобы иметь возможность загрузить какую-либо операционную систему кроме OS/2, IBM включила в состав OS/2 менеджер загрузки (Boot manager). Менеджер загрузки установлен в малый первичный раздел, который отмечен как активный в таблице разделов. То есть, MBR производит запуск менеджера загрузки OS/2, а не загрузку операционной системы. Менеджер загрузки предоставляет пользователю меню, в котором можно выбрать раздел для дальнейшей загрузки. Далее менеджер загрузки производит запуск загрузочного сектора операционной системы по тому же адресу 0x07c0:0x0, что и разделе начальной загрузки.

Загрузочный сектор (Bootsector) содержит блок параметров загрузки (BPB - Boot Parameters Block), представляющий собой структуру, которая содержит некоторые важные параметры, необходимые для загрузки надлежащим образом из раздела. Большинство из них предназначены для дискеты и файловой системы FAT, но имеются параметры, необходимые для загрузки должным образом OS/2. В блоке параметров загрузке есть три важных параметра для загрузки OS/2. А именно: параметр HiddenSectors (Скрытые Сектора), физическое загрузочное устройство и логическое загрузочное устройство. Последний аналогичен имени диска устройства загрузки и крайне важно имя диска с разделом начальной загрузки. Физическое загрузочное устройство - это номер загрузки диска в формате BIOS int 13h: так значение 00h соответствует первой дискете, значение 01h - второй дискете, значение 80h - первому жесткому диску, значение 81h - второму жесткому диску и т.д. Параметр HiddenSectors важен для загрузки OS/2 с логического диска в дополнительном разделе. Для первичных разделов это означает смещение раздела от начала жесткого диска, но для логических дисков это равнозначно количеству секторов в каждой дорожке (63 для современных винчестеров). Параметр HiddenSectors используется для преобразования локального номера сектора (от начала раздела) к общему адресу сектора (от начала жесткого диска). По этой причине многие операционные системы могут загружаться только с основного раздела, подобно Windows или FreeBSD. Но IBM спроектировала OS/2 таким образом, чтобы данная операционная система могла загружаться с логических разделов также как с первичных. Как раз для этой цели менеджер загрузки от IBM устанавливает три вышеупомянутых параметра в загрузочный секторе в виде блока параметров загрузки и только после этого передает управление загрузочному сектору. (Данная особенность нужна менеджеру загрузки для того, чтобы должным образом запустить OS/2 с логического раздела. В данный момент такую возможность предоставляют три менеджера загрузки: IBM Bootmanager, VPart от Veit Kannegieser и AirBoot от Martin Kiewitz).

После передачи управления от MBR или менеджера загрузки, в загрузочный сектор загружается так называемый код blackbox из остальной части загрузочного блока (в HPFS это следующие 15 секторов после загрузочного сектора, и примерно 30 секторов в bootJFS) или из

корневого каталога файла os2boot для FAT. Blackbox является микродрайвером файловой системы (Micro File System Driver - MicroFSD) и содержит несколько функций, позволяющих открывать, читать и закрывать файлы в корневом каталоге загрузочного диска. Также в нем имеется код инициализации и код очистки. Одновременно можно открывать не более одного файла, а в blackbox'ах от IBM могут быть прочитаны только файлы, находящиеся в корневом каталоге.

В ходе инициализации blackbox (MicroFSD), с диска загружаются два файла: os2ldr и os2boot. (os2boot для FAT содержит MicroFSD, но в других файловых системах он содержит код MiniFSD). Файл os2ldr является загрузчиком ядра OS/2. Он не зависит от файловой системы, только для FAT содержит специальный код (вышеупомянутые функции для чтения файлов в разделе FAT, а также для поддержки сброса и восстановления памяти на/с FAT раздел(а)). Помимо этого, os2ldr выполняет функции DosHlp (“помошники” для ядра OS/2), то есть работает наподобие некоторого микроядра (наноядра)- осуществляет некоторые функции, от которых зависит ядро. Также, os2ldr содержит поддержку драйвера устройства OEMHLP$. (Да, OEMHLP$ находится в загрузчике!). Очевидно, что os2ldr представляет нечто большее, чем просто загрузчик операционной системы.

Далее управление от blackbox передается os2ldr, которому предоставляется информация о схеме памяти и точки входа в функции доступа к файловой системе (таким образом, загрузчик получает структуру файловой таблицы, указатель на блок параметров загрузки, структуру физического загрузочного устройства и некоторые флаги). Затем загрузчик перемещает себя на вершину нижней памяти, загружает с диска os2krnl и задействует необходимые устройства (таким образом производится синтаксический анализ формата LX и/или NE, перемещение сегментов в требуемые места, а также исправление адресов функций и переменных для надлежащей связи).

После того управление передается os2krnl, и сведения о памяти вместе с os2boot MiniFSD предоставляются загрузчиком ядру.

Для чтения диска os2krnl использует MiniFSD, а не MicroFSD. MicroFSD предназначен для загрузчика и работает в реальном режиме, а MiniFSD предназначен для ядра и работает в защищенном режиме. MiniFSD имеет формат 16-разрядной NE dll. Его размер ограничен 62 Кб. Процедура sysinit файла os2krnl производит загрузку образа MiniFSD в память. Это происходит на первой стадии системной инициализации при чтении config.sys и загрузке драйверов основных устройств (BASEDEV и PSD).

Перед ядром загружаются дисковые драйверы подсистемы OS/2 (это: ibmlflpy.add, ibmls506.add - дисковые драйверы, фильтр ATAPI для поддержки CDROM'ов, DASD manager (os2dasd.dmd), менеджер томов (os2lvm.dmd)), чтение диска выполняется переключением в реальный режим и вызовом фонкции 13h BIOS. После того как драйверы загружены и инициализированы, ядро использует их для чтения диска, а для доступа к файловой системе продолжает использовать MiniFSD. Это первый этап процесса инициализации системы, как это указано в файле ifs.inf из документации от IBM. На данном этапе ядро использует функции MiniFSD MFS_* для доступа к файловой системе. Эти функции очень похожи на функции MicroFSD, но в отличие от последних, работают в защищенном режиме. На втором этапе ядро связывает MiniFSD в IFS цепочку (в цепочке только IFS), вызывает функцию MFS_TERM для завершения первого этапа, а на втором этапе используются функции FS_* из MiniFSD, так что в этот момент MiniFSD подобна обычной IFS, но при этом поддерживается минимум FS_* функций.

Затем начинается третий этап. Это этап запуска IFS, которая заменяет MiniFSD в IFS цепочке.

После этого возможен полный доступ к файловой системе. Теперь система может читать и записывать файлы на любом диске, за счет поддержки имен диска и возможности открывать одновременно множество файлов. На этом этапе ядро загружает необходимые динамические библиотеки и может загружать драйверы обычных устройств (“device=”).

Далее система продолжает загрузку драйверов устройств, потом обрабатываются инструкции “run=” и “call=”, затем “protshell”, чтобы обеспечить загрузку pmshell.

#### Некоторая информация о процессе загрузки OS/2 PPC

У меня нет системного блока PowerPC, но имеется redbook IBM “Первый взгляд на OS/2 Warp (Издание PowerPC)”, содержащий небольшой раздел относительно последовательности загрузки OS/2 PPC. Этот раздел довольно небольшой и содержит очень немного информации. Также у меня есть файлы конфигурации OS/2 PPC для загрузчика (boot.cfg) и для OS/2 сервера (config.sys). Большая часть драйверов, серверов и библиотек загружается из boot.cfg, а config.sys отвечает за индивидуальные настройки OS/2.

Как сказано в redbook IBM, загрузчик (bl_auto файл в корневом диске) запускается PowerPC ROM непосредственно из раздела начальной загрузки, без какого-либо загрузочного сектора (bootsector), ROM сразу же загружает файл загрузчика. Загрузчик имеет файл конфигурации boot.cfg. В файле конфигурации содержатся имя файла микроядра, начальная задача и другие файлы, которые загрузчик должен загрузить из диска в память. Эти сервисы, загруженные из boot.cfg называются нейтральными от personality сервисами (Personality neutral (PN) services), они независимы от OS/2 Personality и содержат драйверы устройства.

Я не владею никакой информацией относительно того, как начальный загрузчик обращаются к файлам на диске, и если имеется эквивалент MicroFSD в OS/2 PPC, то это подобие использует функции ROM, чтобы читать с диска (аналогично функциям int 13h PC BIOS)

Начальный загрузчик загружает ряд файлов в память, затем запускает микроядро и начальную задачу. Начальная задача вызывает начальную загрузку. Начальный загрузчик передает начальной задаче некоторую информацию наряду с размещнными файлами, которые загружены в память. Начальная задача действует как файловый сервер для других серверов. Иными словами, это обеспечивает доступ к файлам, которые начальный загрузчик загрузил в память. Начальная задача не содержит никаких драйверов устройств, вместо этого имеется доступ к файлам, которые начальный загрузчик загрузил в память. Затем начальная задача осуществляет следующее (цитирую текст от redbook):

1. Загружает Root Name Server
2. Запускает настройки, заданные по умолчанию
3. Запускает постановщик задач
4. Обеспечивает файловые сервисы, которые будут использоваться Task Server
5. Указывает Task Server запустить персональные нейтральные серверы, необходимые для поднятия dominant personality. Персональные нейтральные серверы запускают Регистратор Сообщения, Менеджер Аппаратных ресурсов (Hardware Resource Manager (HRM)), обеспечивают Bus Walkers и драйверы устройства.
6. Запуск Personality.

Начальная задача продолжает до конца своей работы функционировать как файловый сервер.

Затем запускается индивидуальные настройки OS/2. OS/2 personality server анализирует

config.sys и загружает персональные специальные серверы OS/2. Драйверы устройств не относятся к OS/2 personality, поэтому они запускаются с помощью bootstrap и находятся в файле конфигурации начального загрузчика (boot.cfg), а не в файле config.sys.

#### Процесс загрузки микроядра L4. Загрузчик GNU GRUB и спецификация Multiboot

Микроядро L4 может быть запущено как в реальном так и в защищенном режиме. Если запуск произведен в реальном режиме, то переключение в защищенный режим осуществляется отдельно. Подробная процедура загрузки описана в L4 API, справочном описании версии X.2. Для загрузки микроядра L4, обычно используется начальный загрузчик GNU GRUB. GRUB определяет спецификацию multiboot, которая предназначена быть общим протоколом между ядром операционной системы и начальным загрузчиком. В настоящее время только GRUB поддерживает спецификацию Multiboot, но можно создать совместимый начальный загрузчик. Ядра, поддерживающие Multiboot, содержат ядро HURD Mach (GNU GRUB является официальным GNU проектом по разработке начального загрузчика, который был создан специально для проекта GNU HURD. Но он также применим для Linux, FreeBSD, OpenBSD, NetBSD, MacOS X и L4). Но само L4 - не Multiboot ядро, вместо этого, оно использует свой собственный загрузчик/bootstrapper, который в L4Ka:Pistachio представляет собой kickstart, а в L4/Fiasco - rmgr (resource manager).

Спецификация Multiboot требует, чтобы в первых 8192 байтах ядра размещался диспетчер Multiboot. Этот диспетчер определяет требования для загрузчика ядра, такие как: адреса нагрузки различных сегментов ядра, способ инициализации видео и точку входа ядра. Формат исполняемого файла ядра может быть любым, единственное требование - должен иметься диспетчер Multiboot. Но GRUB также непосредственно поддерживает форматы ELF и a.out. (Но в OS/2 (intel) используется формат LX, а в OS/2 PPC - формат ELF).

Начальный загрузчик загружает ядро (в нашем случае это осуществляет kickstart) и ряд дополнительных модулей. Начальный загрузчик загружает ядро и подключает устройства к нему, но модули остаются незадействованными, загрузчик запускает ядро и предоставляет ему указатель на структуру Multiboot. GRUB оставляет ядро в простой среде защищенного режима без возможности листания, допускаемая линия A20 и Регулятор Прерывания остаются неинициализированным. Также, установлен начальный способ видео.

Структура Multiboot содержит информацию относительно схемы памяти и модулей. Для взаимодействия ядра и модулей имеются связи. Эти связи могут использоваться как командные строки для ядра и модулей, а также, как метки, для идентификации модулей.

В L4Ka:Pistachio, kickstart bootstrapper играет роль мультизагрузчика ядра (multiboot kernel). Он получает структуру Multiboot от GRUB, а ELF загружают ядро L4 и начальные серверы. Затем исследуется Kernel Interface Page (KIP) внутри образа L4. После чего, информация от Multiboot предоставляется в структуру Bootinfo, уточненной полем в KIP. Потом kickstart заполняет поля для начального размещения серверов в KIP (начальные серверы - sigma0 и roottask, которые были пропущены через GRUB как мультизагрузочные модули). Затем kickstart вызывает точку входа в ядре L4.

В случае L4/Fiasco роль kickstart играет resource manager (rmgr). Он состоит из двух этапов. 1 этап аналогичен kickstart. Он анализирует конфигурацию и загружает L4 и серверы. 1 этап осуществляет конфигурацию, для обеспечения второго этапа. Во 2 этапе запускается L4, служащее как корневой сервер. Но в настоящее время, rmgr разделен на 2 части - начальная

загрузка(выход на номинальный режим) и roottask, который загружаются как отдельные мультизагрузочные модули.

После запуска L4, перемещает себя к надлежащему месту в памяти, а затем запускаются sigma0 и roottask. После этого, roottask может инициализировать остальную часть системы.

#### История проекта FreeLdr и предложения о путях развития

Проект FreeLdr берет свое начало с 1999 года, когда Дэвид Зиммерли (David Zimmerli) написал статью в EDM/2 (Проект по замене OS2LDR: http://www.edm2.com/0705/freeldr/freeldr.html) относительно os2ldr и начале проекта по его замене. Это была небольшая исполняемая программа формата COM, которая загружалась с помощью blackbox. Она получает информацию от blackbox (структуре файловой таблицы, BPB и т.д.), и использует blackbox, чтобы проверить его функции доступа к файловой системе. FreeLdr выводит полученную информацию на экран и COM порт. Он вызывает функцию Open от blackbox, чтобы показать размер файла os2krnl. Однако, это было сделано только в целях тестирования. Затем, Д. Зиммерли предложил реализовать часть функциональных возможностей os2ldr, но не довел это до конца. Но информация была доступна в статье на странице EDM/2.

Сейчас эти данные взяты проектом osFree, и использованы как начальная точка реализации загрузчика osFree. Название загрузчика осталось прежним, поскольку оно отражает название проекта osFree. Первоначальные источники были написаны в Turbo C/Turbo Assembler. В проекте osFree используются Watcom C и ассемблер, поэтому эти источники были перенесены в OpenWatcom. Портирование осуществляли - Саша Шмидт (Sascha Schmidt) и Юрий Прокушев (Yuri Prokushev). Ими были объединены источники FreeLdr с минимальным набором процедур GRUB, необходимых, для обеспечения загрузки и запуска kickstart. В настоящее время, код связан с функциями GRUB, но не работает. Проблема - с вызываемыми функциями MicroFSD - функция mu_Open () вызывается, но имеется некоторая проблема с прохождением параметров и подтверждениями вызовов. Пока код отлаживается. Мы используем эмулятор Bochs PC и содержащийся в нем отладчик для отладки кода. (Проблема, кажется, простой, но среди нас не было разработчиков, имеющих навыки программирования в ассемблере. И программа отлаживалась только в соответствии с сообщениями отладчика. Теперь когда у нас есть отладчик, возможно, проблема будет вскоре решена ).

В данный момент у меня (Валерия Седлецкого, также известного как valerius) имеются некоторые идеи относительно последовательности загрузки osFree и структуры начального загрузчика. Идеи черпались из последовательности загрузки OS/2 PowerPC и начального загрузчика GRUB. Но главный источник идей, конечно, последовательность загрузки OS/2 (intel). Также, некоторые идеи были заимствованы из os2csm config.sys редактора и препроцессора Вейта Каннегиесера (Veit Kannegieser).

#### Замечание о OS2CSM

Вейт Каннегиесер (Veit Kannegieser) написал программу под названием os2csm. Вдохновением для этого послужила одна DOS-программа, изменяющая config.sys в памяти. Os2csm устанавливается вместо os2ldr, и переименовывает os2ldr в os2ldr.bin. Os2csm сцепляется с программой обработки прерывания int 13h и анализирует информацию, считывая процедуру int 13. Файл Config.sys содержит директивы, аналогичные директивам C препроцессора, и каждые

512 байт файла config.sys (каждый дисковый сектор config.sys) имеют специальный комментарий со специальной сигнатурой в нем. Процедура, которая сцепляется с программой обработки прерывания int 13h, проверяет каждый сектор для этих сигнатур, и если она находит их, то предполагает, что файл config.sys читается. Таким образом, препроцессор определяет и исправляет config.sys на лету.

Препроцессор Config.sys полезен для замены некоторых частей config.sys некоторыми переменными (также известными как символы препроцессора). Эти переменные могут быть получены из меню, которые os2csm предоставляет пользователю. Os2csm тогда загружает os2ldr из файла os2ldr.bin; загрузчик запускает ядро. Когда ядро считывает config.sys, os2csm изменяет его на лету, и заменяет переменные. Так, установочные параметры, установленные пользователем заменяют переменные в config.sys. Таким образом, набор параметров допускается к ядру через препроцессор config.sys.

В настоящее время OS2CSM используется в eCS (в демонстрационном и установочном дисках eCS). Вы можете скачать демонстрационный диск из http://ecomstation.com/.

Мы можем использовать эту идею для нашего начального загрузчика, так, мы сможем реализовать простой препроцессор с синтаксисом similiar в C препроцессоре (или, даже, синтаксис PPWizard). Я предлагаю внедрить этот препроцессор в загрузчик, а вместо сцепления с процедурой int 13, мы можем сцепляться с процедурами blackbox. Конфигурации считываются через blackbox (не через minifsd, подобно в оригинальной OS/2!). Я говорю явно отмечать ряд напильников как конфигурации в конфигурации загрузчика. Так, сигнатуры в каждом секторе файла config.sys не являются необходимыми. Также мы можем использовать более привлекательный и красивый наглядный синтаксис вместо уродливого синтаксиса os2csm.

Директивы препроцессора могут включать аналоги “#define”, “#include”, “#ifdef” и т.д., это позволит определяющим символам включать один файл конфигурации в другой, и условно включать файлы или определяющие символы. Так конечный файл конфигурации может быть гибко создан из частей, а переменные в нем могут быть заменены.

Идея препроцессора дополняет идею GRUB относительно прохождения командных строк к ядру и модулям. То есть, мы предлагаем использовать проходящие параметры к ядру как командные строки moltiboot, также как замену переменных в файлах конфигурации. Таким образом, мы комбинируем эти два подхода.

#### Идеи о дизайне FreeLdr

1) Во-первых, я предлагаю объединить функциональные возможности начального загрузчика OS/2 и os2ldr в одной программе. То есть, последовательность загрузки должна быть в этом роде: MBR нагрузки - активный раздел, или раздел с данным номером. (Я уже написал такой MBR сектор, который может загружать загрузочный сектор из выбранного корневого или логического раздела (да, логические разделы также поддерживаются!). Номер самозагружаемого жесткого диска и номер раздела в нем написаны внутри MBR первого жесткого диска. (Самозагружаемый раздел может находиться на том же самом жестком диске на ином винчестере, с которого производится чтение через MBR).

Так, MBR загружает загрузочный сектор сектор из самозагружаемого раздела. Загрузочный сектор загружает blackbox. Blackbox загружает наш загрузчик. Затем следует запуск загрузчика. Загрузчик объединяет функциональные возможности начального загрузчика с

функциональными возможностями менеджера загрузки: после вызова из blackbox, загрузчик предоставляет меню пользователю. Пользователь выбирает пункт из этого меню, каждый пункт меню определяет операционную систему, которая будет загружена наряду с параметрами, которые затем пропускаются к ядру ОС. После этого пункта, загрузчик/менеджер загрузок готов к чтению загрузочных секторов операционных систем, не поддерживается непосредственно нашим загрузчиком, подобно Windows. Загрузчик только загружает передающий загрузочный сектор и исполняет его. Но если ядро ОС непосредственно поддерживается, то загрузчик может также передавать некоторые параметры к ядру, через файл конфигурации или параметры командной строки.

Преимущество такого подхода состоит в том, что мы можем выбирать операционную систему и ее параметры в одном месте места, а именно в объединенном загрузчике/менеджере загрузки. Например: в оригинальной OS/2 менеджер загрузок позволяет выбирать ОС, а os2ldr позволяет выбирать дополнительные параметры - это осуществляется нажатием выбранной комбинации “горячих клавиш”, чтобы иметь возможность отката к первоначальным значениям. Имеются также множество других параметров, доступные через нажатия Alt-F1 [F2, F3, …]. В нашем случае, имеется только одно меню из которого могут быть выбраны ОС и ее параметры, а также дополнительные меню, подобно выбору глубины отката, - загрузчика/менеджера загрузки. Как видно, преимущество в интеграции возможностей.

Также, наш менеджер загрузок располагается на обычном разделе OS/2, а не на специальный разделе. Для чтения файлов он использует microfsd. И все установочные параметры менеджера загрузки/начального загрузчика могут быть сохранены в обычных текстовых файлах конфигурации.

И наконец, наша идея относительно гибрида загрузчика/менеджера загрузок позволяет нам загружать ядра отличные от OS/2 из того же раздела. И не только ядро, любая версия системной составляющей может быть выбрана от того же самого места - они могут быть выбраны через меню менеджера загрузок.

2) Загрузчик предоставляет меню пользователю. Каждый пункт меню соответствует сценарию загрузки. Сценарий содержит команды, для получения дополнительной информации от пользователя (то есть, эта команда показывает меню, пользователь изменяет параметры, после чего параметры возвращаются загрузчику. Затем параметры составляют загрузчик “enviroment”. Нити enviroment могут заменять переменные в командных строках и файлах конфигурации, изменять текущий раздел, определять переменные и т.д. Также сценарий загрузки содержит определение файлов, загруженных начальным загрузчиком. Загрузчик различает исполняемые файлы (загрузчик выполняет синтаксический анализ формата), файлы, которые только загружаются загрузчиком, но их формат им не анализируется, и файлы конфигурации. Файлы, отмеченные как конфигурационные - выполняются препроцессором.

Так, далее следуют файлы конфигурации: I) файл определения меню загрузчика, он содержит структуру пунктов меню. Каждый пункт меню связывается с ним через сценарий загрузчика. Этот файл конфигурации подобен файлу menu.lst в GRUB. II) сценарии загрузки. Каждый сценарий упомянут или включен конфигурацией определения меню. Каждый сценарий подобен файлу boot.cfg в OS/2 PPC. III) файл config.sys. ОН относится к OS/2 personality. Эти конфигурации читаются через запросы microfsd и могут быть исполняемы. Могут иметься дополнительные файлы конфигурации для отдельных серверов. Они также используют средства предварительной обработки конфигурации загрузчика, так и параметры, определенные в сценарии загрузчика, запрашивают у пользователя, можно заменять переменные в этих файлах конфигурации. Так что мы можем гибко устанавливать параметры каждого системного компонента в меню менеджера загрузок.

Также, для лучшей гибкости, мы можем осуществлять небольшую конфигурацию для blackbox. Когда blackbox запущен загрузочным сектором, он может читать файл конфигурации, из которого становится известно, какие файлы он должен загрузить как minifsd (это необязательно, мы не можем использовать minifsd, так что нет необходимости загружать его) и главный модуль freeldr. (См. следующий параграф: идея относительно модулей).

3) Идея относительно модулей.

В настоящее время, загрузчик - COM-файл, ограничен размером в 64 Кб. Чтобы включать большее количество функциональных возможностей, может быть имеет смысл осуществлять данные задачи с помощью мультисегментной программы, так что, должен использоваться лучший формат EXE. Поскольку это - 16-разрядная программа реального режима, и, вероятно, формат OS/2 NE подходит для этого. Формат DOS EXE, кажется, наиболее простой, так что более просто реализовать FreeLdr как файл DOS EXE. Под модульным загрузчиком (иметь возможность загрузить только необходимые его части, и возможности загрузить/разгрузить их же в любое время), я имею в виду реализовать его как набор модулей. Модуль должен работать в реальном режиме и может быть реализован как файл DOS EXE. (Мы не можем использовать DLL'и в реальном режиме, так что мы должны проектировать простой механизм, основанный на файлах DOS EXE). Я предлагаю модуль, представляющий из себя DOS EXE файл с с заголовочным файлом (*.h). Заголовочный файл помогает располагать функций внутри EXE файла. Заголовочный файл начинается с указателя на нить ASCIIZ, которая содержит название модуля. После этого указателя следует размер диспетчера, затем размер выполнимой программы DOS после заголовочного файла и затем следует таблица структур, которая может быть описана как:

```
struct {
  char *FuncName;
  unsigned long EntryPoint;
} *pFuncTable;
```

то есть, каждая структура определяет функцию в EXE файле, это связывает имя функции с его смещением в EXE файле. Эта матрица структур сопровождается таблицей нити, которая содержит все имена функции и модульное название. Каждый FuncName указатель указывает на значение в этой таблице нити. Заголовочный файл помогает располагать каждую функцию в EXE файле. Функциональная таблица может быть сгенерирована из файла карты компоновщика.

Главный модуль загрузчика загружается с помощью microfsd. Он имеет формат DOS EXE, для его исполнения, мы должны загрузить его некоторым выполнимым загрузчиком формата. DOS EXE загрузчик может быть реализован как DOS COM файл. Я имею в виду связать DOS EXE загрузчик с главным модулем FreeLdr (главный модуль скреплен с выводом EXE загрузчика, эта идея заимствована из MS NTLDR: NTLDR состоит из запускаемого COM файла в связке с исполняемой программой формата PE). После запуска FreeLdr получает информацию от blackbox, загружает и перемещает главный модуль из его вывода, выполняет и пропускает информацию, полученую от blackbox.

Главный модуль содержит механизм для загрузки других модулей из файлов, расположенных на диске. Он загружает модуль как DOS EXE файл, и связывает его заголовочный файл со списком заголовочных файлов. При выполнении настроек модуля, загрузчик формата DOS EXE исправляет адреса в заголовочных файлах, которые связаны со списком. Из этого списка главный модуль может размещать любае функции от любого модуля. Для той цели, главный

модуль поставляет достает функцию, которую нужно назвать от других модулей. Эта функция берет модульное имя, а имя функции в ней служат как параметры, и возвращает точку входа этой функции. Этим путем, любой модуль может находить точку входа к любой функции с данным именем в любом другом модуле с данным именем. Так, мы имеем своего рода службу имен, которая может преобразовывать имя функции к его адресу. Любой модуль может вызывать любую функцию в другом модуле.

Microfsd, загрузчики для различных форматов файла (DOS EXE, NE, LX, ELF), препроцессор файла конфигурации и т.д. могут быть реализованы как отдельные модули.

4) Мы можем осуществлять дополнительные исполняемые загрузчики форматов для форматов, отличных от ELF (Например, LX, NE, PE (?) ...). Они также могут быть реализованы как отдельные модули.

5) Возможность читать файлы из другого раздела, а не только в загрузочном, можно осуществить переключением blackbox. Для этого, загрузчик может выполнить команду сценария загрузки, изменять текущий диск, подобно команде “root” в GRUB. Для этого, загрузчик загружает новый microfsd для измененного раздела файловой системы. Он обновляет структуру файловой таблицы указателями на функции в новом microfsd. Он также загружает BPB из нового раздела загрузочного сектора (и исправляет значение HiddenSectors (Скрытые Секторы), при необходимости). Так, эта особенность может давать возможность загрузчику читать файлы из нескольких разделов, переключая их.

6) Я предлагаю сделать загрузчик способный загружать стандартные мультизагрузочные ядра, ядро L4 (как своего рода мультизагрузочное ядро), и отдельные ядрами, типа ядра OS/2. Ранее, эта идея уже описывалась в данном тексте. Идея состоит в том, чтобы реализовать загрузчик в виде набора загружаемых модулей. Отдельные ядра ОС, не совместимые со спецификацией мультизагрузки, могут быть поддержаны отдельным загрузочным модулем. Поддержка мультизагрузки может также быть реализована как отдельный модуль. Модуль загружается главным модулем Freeldr. Написанием модуля поддержки для отдельного типа ядра сторонние разработчики могут дополнить наш загрузчик возможностью загружать их ядра. Может статься, что наш загрузчик удовлетворит не только ядра UNIX, OS/2 или L4, но также и ядра Windows и ReactOS. Мы не можем ждать дядю Билли, чтобы сделать поддержку загрузки Windows ядра нашим загрузчиком, но возможно, что парни, занимающиеся проектом ReactOS, смогут сделать свое ядро загружаемым с помощью FreeLdr . Та часть загрузчика, которая загружается blackbox называется общей частью загрузчика. Общая часть содержит только функции поддержки для загрузки модулей, расположение в них функций, загрузчик формата DOS EXE, и несколько других функций. Это дает возможность контролировать заказную часть вместе с интерфейсами в модульном загрузчике, microfsd, а также информацию, полученную от blackbox самозагружаемого раздела. Отдельная часть загрузчика осуществляет поддержку загрузки определенного вида ядра.

а) Для загрузки мультизагрузочных ядер, специальная мультизагрузочная часть загрузчика может быть реализована как отдельный модуль. посредством этого, мы можем загружать L4 ядро, также как большинство UNIX ядер.

б) Для загрузки обычных OS/2 ядер, должна иметься специально предназначенная часть. Эта особая часть берет от общей части информацию, полученную от blackbox. Она загружает файл os2ldr с диска и передает эту информацию os2ldr. Далее процесс загрузки продолжается как обычно. В будущем, особая часть может быть расширена, так что она полностью заменит функциональные возможности os2ldr, согласно идее Дэвида Зиммерли, состоящая в том, чтобы

в точности заменить функциональные возможности os2ldr.

в) При загрузке неподдерживаемых ядер, загрузчик может только загружать передаваемые загрузочным сектором ОС, и передавать им управление. Это может быть реализовано подобно команде GRUB “chainloader”.

#### Предложенная последовательность загрузки

1) Что нам нужно в MiniFSD?

Если мы посмотрим на последовательность загрузки OS/2 PPC и L4, то мы можем придти к выводу, что начальный загрузчик OS/2 PPC и GRUB осуществляют одинаковые функции. Они загружают ядро и набор файлов в память и запускают ядро. Затем задача начальной загрузки, в 1-ом случае, и root task в 2-ом случае, запустит другие задачи; FreeLdr также должен выполнять эквивалентные задачи. В этот момент файловая система еще не инициализирована, bootstrap, или root task может только обращаться к файлам, которые были уже загружены начальным загрузчиком. Чтобы использовать файловую систему, мы должны сначала загрузить дисковый драйвер (ibm1s506.add или ibm1flpy.add), dasd администратор, volume администратор и драйвер файловой системы (или их эквиваленты в нашей микроядерной системе). Возможно, что мы сможем использовать minifsd как драйвер файловой системы. В случае с L4, мы не можем использовать 16-разрядные программы, таких же простых как в оригинал OS/2. Таким образом, наш minifsd должен быть 32-разрядным. И ограничение в 62 Кбайта для его размера не применимо в данном случае, поскольку мы используем 32-разрядные программы.

Как сказано в ifs.inf файле от IBM, minifsd имеет два режима работы - в стадии 1 и в стадии 2 во время процесса загрузки. Прежде началом стадии 1, когда minifsd инициализируется, он не может вызывать никакие запросы dynalink (в MFS_INIT процедуре инициализации). Обычная IFS в FS_INIT процедуре, может вызывать внешние dlls. Единственные внешние функции, которые minifsd может вызывать - MFSH_* helpers, которые вызываются из ядра. Полноценные IFS могут осуществлять намного большее количество внешних вызовов. Это - FSH_* IFS helpers. (Но, насколько я понял, IFS не может вызывать другие внешние функции, кроме FSH_* вызываются процедуры отличные от процедуры FS_INIT). Чтобы иметь возможность вызывать DLLS в IFS init time, поддержка динамической загрузки должна работать и оперативно, а CAM DLLS должны быть доступны. Они могут быть загружены только minifsd (потому что IFS все же не инициализирована), или же их можно запускать начальным загрузчиком.

Причина, почему minifsd используется в процессе загрузки OS/2 (intel), состоит в том, чтобы иметь ограниченный доступ к файловой системе после переключения ядра в защищенный режим. Прежде, чем дисковые драйверы подсистемы OS/2 будут загружены, дисковое чтение выполняется при временном переключении в реальный режим и вызове функции чтения диска int 13h. После того, как дисковая подсистема загружена, чтение с диска выполняется через дисковый драйвер OS/2.

В L4 или в OS/2 PPC, прежде чем дисковые драйверы будут загружены, мы не можем переключить в реальный режим, чтобы назвать процедуры int 13. Следовательно, дисковые драйверы должны быть считаны microfsd с помощью начального загрузчика, а затем их нужно направить к roottask через память. Таким образом, дисковые драйверы могут быть запущены из памяти с помощью roottask. Но тогда спрашивается, почему мы должны использовать minifsd? Мы сможем загружать сразу полноценный IFS, а перед этим, мы сможем запускать

динамические серверы поддержки загрузчика в формате ELF и загружать весь набор DLLS, необходимых IFS. Все эти файлы можно пропускать начальным загрузчиком наряду с дисковыми драйверами подсистемы.

Так что нет особого резона использовать minifsd в последовательности загрузки L4, необходимы только microfsd. И если мы смотрим на OS/2 PPC, то мы увидим, что она не имеет minifsd и загрузку фазы basedev. Вместо этого, все требуемые сервисы предоставляются начальным загрузчиком задаче начальной загрузки. В нашем случае, FreeLdr и kickstart играют роль начального загрузчика OS/2, а roottask аналогична задаче начальной загрузки. В общем, мы должны, использовать подобные решения при проектировании и разработке.

2) В случае загрузки L4, загрузчик загружает kickstart L4 bootstrapper. Kickstart'у начальным загрузчиком предоставляется набор модулей через структуру мультизагрузки. Kickstart затем передает эту информацию в структуре bootinfo, направленной полем в KIP. Когда L4 запущено, оно запускает sigma0 и roottask. Roottask может получить информацию от структуры bootinfo, которая может быть взята из KIP. Адрес KIP может быть получен через L4 системный вызов KernelInterface(). Так, roottask может получить информацию относительно модулей, полученных от kickstart. Тогда roottask может найти необходимые модули в структуре bootinfo. Цель каждого модуля может быть определена через нити, связанные с модулями, каждый модуль может быть отмечен специальной меткой, которая определяет ее назначение (например, модуль, содержащий библиотеку, конфигурацию, сервер root name, сервер загрузrb исполняемых файлов и т.д.). Метка может содержаться в нити наряду с командной строкой для этого модуля. Данным способом, информация относительно модулей пропускется от загрузчика через kickstart к roottask, а roottask может находить необходимые серверы и загружать их в надлежащий порядок. Таким образом, roottask может осуществлять функции задачи начальной загрузки OS/2 PPC. Roottask сначала запускает персональные нейтральные сервисы (personality neutral (PN) servuces), затем осуществляет персональные настройки ОС (OS/2, L4Linux и т.д.).

#### Обзор последовательности загрузки osFree (Нормативный)

При включении компьютера или его перезапуске, первая программа, которая будет запущена - это БАЗОВАЯ СИСТЕМА ВВОДА-ВЫВОДА (BIOS). Существуют различные базовые системы ввода-вывода с различной последовательностью выполнения. Мы должны знать только одну вещь: БАЗОВАЯ СИСТЕМА ВВОДА-ВЫВОДА (Standart, BOOTrom, PXE-БИО или какая-либо ещё) загружает сектор начальной загрузки (под термином ‘сектор начальной загрузки’, мы понимаем не только фактический сектор начальной загрузки жесткого диска или образ PXE загрузки, но и любой первый код, который загружен и выполнен аппаратными средствами) и передает ему управление. И тут запускается наша последовательность загрузки. Мы называем весь код, выполненный перед нашим сектором начальной загрузки как BlackBox (Черный Ящик). Мы не знаем, как он работает. Мы только знаем, что у нас есть возможность передачи управления нашему коду. Наш сектор начальной загрузки представляет собой 16-разрядный код. Сектор начальной загрузки загружает загрузчик первого этапа, именуемый MicroFSD, MiniFSD и Загрузчик ядра. MicroFSD - Микро Драйвер Файловой системы. MiniFSD представляет собой Малый Драйвер Файловой Системы. Загрузчик ядра - код, который загружает и выполняет ядро osFree (или любое мультизагрузочное совместимое ядро, которое предпочитает пользователь).

Код в секторе начальной загрузки заполняет информационные структуры и передает их загрузчику ядра. Те информационные структуры, которые содержат информацию

относительно текущего распределения памяти и точек входа MicroFSD.

Загрузчик ядра - это смешанный 16/32-рязрядный код. Он загружает совместимый образ мультизагрузочного ядра, перераспределяет его и MiniFSD в памяти, точках входа связей), переключает процессор в защищенный режим и передает управление ядру. Ядро и MiniFSD представляют собой уже не 16-разрядный, а 32-разрядный код.

### Интерфейс MicroFSD/KernelLoader (Нормативный)

Интерфейс MicroFSD/KernelLoader - такой же что и у OS/2. После того как MicroFSD загрузил весь требуемый код (образ MiniFS, образ OS3LDR), управление передается загрузчику ядра (OS3LDR). Процессор должен быть в реальном режиме, а регистры процессора должны быть заполнены соответственно следующей таблице.

При первоначальной передаче управления OS3LDR от “черного ящика” (“black box”), интерфейс определяется следующим образом:

| Register | Contains | Description |
|---|---|---|
| DH | Boot mode flags | bit 0 (NOVOLIO) on indicates that the mini- FSD does not use MFSH_DOVOLIO. |
| | | bit 1 (RIPL) on indicates that boot volume is not local (RIPL boot) |
| | | bit 2 (MINIFSD) on indicates that a mini-FSD ispresent. |
| | | bit 3 (RESERVED) |
| | | bit 4 (MICROFSD) on indicates that a micro-FSD is present. |
| | | bits 5-7 are reserved and MUST be zero. |
| DL | Boot disk drive number | This parameter is ignored if either the NOVOLIO or MINIFSD bits are zero. |
| DS:SI | pointer to the BOOT Media's BPB | This parameter is ignored if either the NOVOLIO or MINIFSD bits are zero. |
| ES:DI | pointer to a filetable structure | |
| | | /* # of entries in this table */ |
| | | unsigned short ft_cfiles; |
| | | /* paragraph # where OS2LDR is loaded */ |
| | | unsigned short ft_ldrseg; |
| | | /* length of OS2LDR in bytes */ |
| | | unsigned long ft_ldrlen; |
| | | /* paragraph # where microFSD is loaded */ |
| | | unsigned short ft_museg; |
| | | /* length of microFSD in bytes */ |
| | | unsigned long ft_mulen; |
| | | /* paragraph # where miniFSD is loaded */ |
| | | unsigned short ft_mfsseg; |
| | | /* length of miniFSD in bytes */ |
| | | unsigned long ft_mfslen; |
| | | /* paragraph # where RIPL data is loaded */ |

| Register | Contains | Description |
|---|---|---|
| | | unsigned short ft_ripseg; |
| | | /* length of RIPL data in bytes. */ |
| | | unsigned long ft_riplen; </pre> |

```
/* The next four elements are pointers to
    microFSD entry points     */
 unsigned short(far *ft_muOpen)(char far *pName,
                unsigned long far *pulFileSize);
 unsigned long (far *ft_muRead)(long loffseek,
           char far *pBuf, unsigned long cbBuf);
 unsigned long (far *ft_muClose)(void);
 unsigned long (far *ft_muTerminate)(void);
```

Точки входа интерфейса MicroFSD определены следующим образом:

<ul> <li class="level1"> Mu_Open передает удаленный указатель на имя файла, который будет открыт и удаленный указатель на ULONG, для возврата размера файла. Возвращаемое значение (в AX) означает успех (0) или неисправность (не -0).  </li> <li class="level1"> Mu_Read передает искомое смещение, удаленный указатель на буфер данных, и размер буфера данных. Возвращаемое значение (в DX:AX) указывает число фактически читаемых байтов.  </li>

<li class="level1"> Mu_Close не имеет никакие параметры и не ожидает никакого возвращаемого значения. Это - сигнал для micro-FSD, что загрузчик прочитывает текущий файл.  </li> <li class="level1"> Mu_Terminate не имеет никаких параметров и не ожидает никакого возвращаемого значения. Это - сигнал для micro-FSD, что загрузчик закончил чтение загрузочного диска. </li> </ul>

Загрузчик вызовет micro-FSD в последовательности Открыть-Читать-Читать-....-Читать-Закрыть для каждого считываемого файла с загрузочного диска. После того, как все файлы загружены, должен быть вызван mu_Terminate. 

<h2><a name="интерфейс_kernelloader_kernel_нормативный" id="интерфейс_kernelloader_kernel_нормативный">Интерфейс KernelLoader/Kernel (Нормативный)</a></h2> 

 Интерфейс KernelLoader/Kernel не совместим с <acronym title="Operating System">OS</acronym>/2, но совместим с мультизагрузкой. Это означает, что Вы можете загружать различные ядра, например Linux-ядро. 

 Имеются три основных аспекта образа интерфейса Kernel loader/Kernel:

 <ol> <li class="level1"> Формат образа ядра как омечаемый загрузчиком ядра (Kernel loader).

</li> <li class="level1"> Состояние машины, когда загрузчик ядра запускает операционную систему. </li> <li class="level1"> Формат информации,

передаваемой загрузчиком ядра операционной системе. </li> </ol>

<h3><a name="формат_образа_ядра" id="формат_образа_ядра">Формат образа ядра</a></h3> 

Образ ядра может представлять собой обычный 32-разрядный исполняемый файл в стандартном формате для какой-либо особенной операционной системы, за исключением того, что он может быть связан с загружаемым по неумолчанию адресом, чтобы избежать сверхзагрузки области Ввода - вывода PC или других резервных областей, и конечно не должны использоваться общедоступные библиотеки или другие причудливые особенности. 

 Образ ядра должен содержать дополнительный заголовочный файл, называемый заголовочным файлом мультизагрузки, помимо формата заголовочных файлов, используемого образом ядра. Заголовочный файл мультизагрузки должен полностью находиться в пределах первых 8192 байтов образа ядра, и должен быть 32-разрядным. Вообще, он должно прибыть как можно раньше и может быть внедрен в начало текстового сегмента после реально выполненного заголовоного файла. 

<h3><a name="структура_заголовка_multiboot" id="структура_заголовка_multiboot">Структура заголовка Multiboot</a></h3> 

Схема заголовочного файла Мультизагрузки должна быть следующей: 

 Смещение Тип Название поля Примечание 0 u32 Magic требуемый 4 u32 Флаги требуемый 8 u32 Контрольная сумма требуемый 12 u32 Header_addr , если флаги [16] установлены 16 u32 Load_addr , если флаги [16] установлены 20 u32 Load_end_addr , если флаги [16] установлены 24 u32 Bss_end_addr , если флаги [16] установлены 28 u32 Entry_addr , если флаги [16] установлены 32 u32 Mode_type должно игнорироваться 36 u32 Ширина должно игнорироваться 40 u32 Высота должно игнорироваться 44 u32 Глубина должно игнорироваться Поля magic, флаги и контрольная сумма определены в полях magic заголовочного файла, области header_addr, load_addr, load_end_addr, bss_end_addr и entry_addr определены в полях адреса заголовочного файла, а поля mode_type, ширина, высота и глубина - определены в полях графики заголовочного файла. 

 Поля magic заголовочного файла мультизагрузки, поле magic является чудесным номером, опознающим заголовочный файл, который должен быть шестнадцатеричное значением 0x1BADB002. 

 Помечание полевых флагов определяет особенности, которые образ ядра запрашивает или требует от загрузчика ядра. Биты 0-15 указывают требования; если загрузчик ядра видит любой бит из этого набора, но не понимает флаг или не может выполнять требования, он указывает по каким причинам, затем должен уведомить об этом пользователя и прекратить дальнейшую загрузку образа ядра. Биты 16-31 указывают необязательные особенности; если

любые биты в этом диапазоне установлены, но загрузчик ядра не понимает их, он может просто игнорировать их и работать как обычно. Естественно, все пока еще -неопределенные биты в слове флагов должны быть установлены на нуль в образах ядра. Таким образом, поля флагов служат для управления версией как для простого выбора особенности. 

 Если бит 0 в слове флагов установлен, то все загрузочные модули, загруженные наряду с операционной системой должны быть выровнены на границах страницы (4Кб). Некоторые операционные системы могут быть способны отображать страницы, содержащие загрузочные модули непосредственно в изменяющихся страницах адресного пространства во время запуска, и следовательно, нуждаться в загрузочных модулях, чтобы преобразвать в выровненную страницу. Если бит 1 в слове флагов установлен, то информация относительно доступной памяти,по крайней мере, через mem_* поля структуры информации мультизагрузки (см. формат информации загрузки) должен быть доступна. Если загрузчик ядра способен к принятию карты памяти (mmap_* поля) и она имеется, то данное поле может быть также включено. Бит 2 в слове флагов должен игнорироваться загрузчиком ядра. Если бит 16 в слове флагов установлен, то поля в смещениях 8-24 в заголовочном файле мультизагрузки допустимы, и ядерный загрузчик должен использовать их вместо полей в фактическом исполняемом заголовочном файле, чтобы определить, где загрузить образ ядра. Этой информации не должно быть, если образ ядра находится в формате ELF, но она должна быть, если образ находится в формате a.out или в некотором другом формате. Соответствующие загрузчики ядра должны быть способны загрузить образы, которые или находятся в формате ELF, или содержат нагрузку, адресуют информацию, внедренную в заголовочный фйал мультизагрузки; они могут также непосредственно поддерживать другие выполнимые форматы, в частности a.out варианты, но их поддержка не является необходимой.

 Поле контрольная сумма является 32-разрядным значением без знака которое, будучи добавленным к другим полям magic (то есть magic и флаги), должно иметь 32-разрядную сумму отличную от нуля. 

 Поля адреса заголовочного файла мультизыгрузки. Все поля адреса, допускаемые битом 16 флага - физические адреса. Значение каждого следующие: 

 Header_addr содержит адрес, соответствующий началу заголовочного файла мультизагрузки - физическое размещение памяти, в котором magic значение, как предполагается, будет загружено. Это поле служит, чтобы синхронизировать картографирование между смещениями образа ядра и физическими адресами памяти. Load_addr содержит физический адрес начала текстового сегмента. Смещение в файле образа ядра, для начала запуска загрузки, в котором заголовочный файл был найден, минус (header_addr - load_addr). Load_addr должен быть меньше или равным header_addr. Load_end_addr содержит физический адрес конца сегмента данных. (Load_end_addr - load_addr) определяет сколько данных требуется загрузить. Это подразумевает, что текст и сегменты данных должны быть последовательны в образе ядра; это касается существующих a.out выполнимых форматов. Если это поле нулевое, загрузчик ядра предполагает, что текст и сегменты данных занимают целый файл образа ядра. Bss_end_addr содержит физический адрес конца сегмента bss. Загрузчик ядра инициализирует эту область, чтобы установить на нуль, и резервирует память, которую она занимает, чтобы избежать размещения модуля загрузки и других данных, в области предназначенные операционной системе. Если это поле нулевое, загрузчик ядра предполагает, что bss сегменты отсутствуют. Entry_addr физический адрес, к которому должен перейти загрузчик ядра, для того, чтобы операционная система начала функционировать. Состояние машины, когда загрузчик ядра вызывает 32-разрядную

операционную систему, должно иметь следующий вид: 

 EAX должен содержать значение magic; присутствие этого значения указывает на операционную систему, что она была загружена мультизагрузочным загрузчиком ядра (например, в противоположность другому типу ядерного загрузчика, которым операционная система может также быть загружена). EBX должен содержать 32-разрядный физический адрес структуры информации мультизагрузки, обеспеченной загрузчиком ядра (см. формат информации загрузки). CS должна быть 32-разрядным сегментом кода читать/выполнять со смещением 0 и пределом 0xFFFFFFFF. Точное значение неопределено. DS/ES/FS/GS/SS должен быть 32-разрядными сегментами данных для чтения/записи со смещением 0 и пределом 0xFFFFFFFF. Точные значения неопределены. Гейт A20 должен быть запущен. CR0 Bit 31 (PG) должен быть очищен. Бит 0 (PE) должен быть установлен. Другие биты неопределены. EFLAGS Bit 17 (VM) должен быть очищен. Бит 9 (IF) должен быть очищен. Другие биты неопределены. 

 Все другие регистры процессора и биты флага неопределены. В особенности это касается: 

 Образ ядра ESP должен создать свой собственный набор, как только он потребуется. GDTR Даже при том, что регистры сегмента установлены как описано выше, GDTR, может быть недопустим, так что образ ядра не должен загружать никаких регистров сегмента (даже только перезагружающий те же самые значения!) пока не будет установлен его собственный GDT. Образ ядра IDTR должен оставить выведенные из строя прерывания, пока не будет установлен его собственный IDT. 

 Однако, помимо этого машина должна быть установлена загрузчиком ядра в нормальное рабочее состояние, то есть как инициализировано BIOS (или DOS, в зависимомти от конфигурации). Другими словами, операционная система должна быть способна делать запросы BIOS даже после загрузки, пока она не записывает поверх структур данных BIOS перед выполнением. Также, загрузчик ядра должен оставить PIC, запрограммированный с нормальными значениями BIOS/DOS, даже если они изменены в 32-разрядную модификацию. 

 При загрузке информации формата Upon в операционную систему, регистр EBX содержит физический адрес структуры данных информации мультизагрузки, через которую загрузчик ядра передает необходимую информацию операционной системе. Операционная система может использовать или игнорировать любые части структуры, котрую она выбирает; вся информация, которую передает загрузчик ядра является консультативной. 

 Структура информации мультизагрузки и связанные с ней подструктуры могут быть размещены где-нибудь в памяти загрузчиком ядра (конечно, за исключением памяти, сохраненной для ядра и загрузочных модулей). Таким образом, операционная система способна избежать записи поверх этой памяти. 

 Формат структуры информации мультизагрузки (на данный момент) представляет собой: 

 0 Flags (Требуемый) 4 Mem_lower (Задействован, если флаги [0] установлены) 8 Mem_upper (Задействован, если флаги [0] установлены) 12 Boot_device (Задействован, если флаги [1] установлены) 16 Cmdline (Задействован, если флаги [2] установлены) 20 Mods_count (Задействован, если флаги [3] установлены) 24 Mods_addr (Задействован, если флаги [3] установлены) 28 - 40 Syms (Задействован, если флаги [4] или флаги [5] установлены) 44

Mmap_length (Задействован, если флаги [6] установлены) 48 Mmap_addr (Задействован, если флаги [6] установлены) 52 Drives_length (Задействован, если флаги [7] установлены) 56 Drives_addr (Задействован, если флаги [7] установлены) 60 Config_table (Задействован, если флаги [8] установлены) 64 Boot_loader_name(Задействован, если флаги [9] установлены) 68 Apm_table (Задействован, если флаги [10] установлены) 72 Vbe_control_info(Должен быть заполнен) 76 Vbe_mode_info 80 Vbe_mode 82 Vbe_interface_seg 84 Vbe_interface_off 86 Vbe_interface_len Первый longword указывает присутствие и законность других полей в структуре информации мультизагрузки. Все неопределенные биты должны быть обнулены загрузчиком ядра. Любой набор битов, которые операционная система не понимает, должен игнорироваться. Таким образом, поле флагов также функционирует как индикатор версии, позволяя структуре информации мультизагрузки быть расширенной в будущем без ломки или нарушения чего-либо. 

 Если бит 0 установлен в параметре flags, то mem_* поля правильны. Mem_lower и mem_upper указывают количество более низких и область старших адресов, соответственно, в килобайтах. Более низкая память начинается в адресе 0, и запусках области старших адресов в адресе 1 мегабайт. Максимальное возможное значение для более низкой памяти - 640 килобайт. Значение, возвращенное для области старших адресов - максимальное, адрес первого отверстия области старших адресов минус 1 мегабайт. Это не значит, что оно будет этим значением. 

Если бит 1 в параметре flags установлен, то поле boot_device правильно, и указывает, от которого дискового устройства BIOS загрузчик ядра загрузил образ ядра. Если образ ядра) не был загружено с дискового устройства BIOS, то это поле не должно присутствовать (бит 3 должен быть чист). Операционная система может использовать это поле как подсказку для определения ее собственного корневого раздела, но оно необязательно. поле Boot_device размещено в четырех подполях одним байтом следующим образом: 

 Диск Раздел1 Раздел2 Раздел3  <pre class=“code”> Первый байт содержит номер диска BIOS как понято BIOS INT 0x13 низкоуровневого дискового интерфейса: например 0x00 для первого гибкого диска или 0x80 для первого жесткого диска.</pre>

Три остающихся байта определяют раздел начальной загрузки. Part1 определяет номер раздела верхнего уровня, part2 определяет подраздел в разделе верхнего уровня, и т.д. Нумерация разделов всегда начинаются с нуля. Неиспользованные байты раздела должны быть установлены в 0xFF. Например, если диск разбит на разделы, используя простой одноуровневый DOS, выделяющий разделы схемы, то part1 содержит номер раздела DOS, а part2, и part3 - 0xFF. Другой пример, если диск разбит на разделы сначала в DOS PARTITION, и затем, один из тех DOS PARTITION подразделен в несколько BSD разделы, использующие disklabel стратегию BSD'S, то part1 содержит номер DOS PARTITION, part2 содержит подраздел BSD в пределах того DOS PARTITION, а part3 - 0xFF. 

В DOS разделы обозначены как числа раздела, начинающиеся от 4 и выше, скорее как вложенные подразделы, даже при том, что основная схема диска расширенных разделов по характеру иерархическая. Например, если загрузчик ядра загружается из второго раздела на диске, разбитом на разделы в обычном стиле DOS, то part1 будет 5, и part2, и part3 будут 0xFF.

 Если бит 2 флагов longword установлен, поле cmdline допустимо и содержит физический адрес командной строки, которую нужно пропустить к ядру. Командная строка - законченная нулем строка в C-стиле. 

 Если бит 3 флагов установлен, то поля mods указывают ядру какие загрузочные модули были загружены наряду с образом ядра), и где они могут быть найдены. Mods_count содержит число загруженных модулей; mods_addr содержит физический адрес первой модульной структуры. Mods_count может быть нулевой, не указывая какие загрузочные модули были загружены, даже если бит 1 флагов установлен. Каждая модульная структура форматируется следующим образом: 

 0 Mod_start 4 Mod_end 8 String 12 reserved(0) Первые два поля содержат начало и конечные адреса непосредственно загрузочных модулей. Поле строки (String) обеспечивает произвольную нить(шпагат), которая будет связана с тем специфическим загрузочным модулем; это - законченная нулем сторока <acronym title="American Standard Code for Information Interchange">ASCII</acronym>, точно так же как командная строка ядра. Поле строки может быть 0, если не имеется никакой строки, связанной с модулем. Обычно строка может быть командной строкой (например, если операционная система обрабатывает загрузочные модули как исполняемые программы), или имя пути (например, если операционная система обрабатывает загрузочные модули как файлы в файловой системе), но его точное использование предназначено для операционной системы. Сохраненное поле должно быть установлено в 0 загрузчиком ядра и игнорироваться операционной системой. 

 Внимание: биты 4 и 5 взаимно исключаемы. 

 Если бит 4 в слове флагов установлен, то допустимы следующие поля в структуре информации мультизагрузки, начинающейся в байте 28: 

 28 Tabsize 32 Strsize 36 Addr 40 reserved (0) Они указывают, где таблица идентификаторов от a.out образа ядра может быть найдена. Addr - физический адрес размера (4-байтовый без знака длинный) матрицы a.out, форматируют структуры nlist, сопровождаемые немедленно матрицой непосредственно, тогда размер (4-байтовый без знака длинный) набора законченных нулем строк <acronym title="American Standard Code for Information Interchange">ASCII</acronym> (плюс sizeof (без знака длинный) в этом случае), и наконец, самостоятельный набор строк. Tabsize равен его параметру размера (найденный в начале секции символа), а strsize равен его параметру размера (найденный в начале секции строки) следующей таблицы строки, к которой обращается таблица идентификаторов. Обратите внимание, что tabsize может быть 0, не указывая никакие символы, даже если бит 4 в слове флагов установлен. 

 Если бит 5 в слове флагов установлен, то допустимы следующие поля в структуре информации мультизагрузки, начинающейся в байте 28:

 28 Num 32 Size 36 Addr 40 Shndx Где указывается секция таблицы заголовочного файла из ELF ядра, размер каждого входа, число входов, и строка таблицы, используемой как индекс имен. Они передают shdr_* входам (shdr_num, и т.д.) в выполнимом и пригодном для редактирования формате спецификацию (ELF) в заголовочном файле программы. Все секции

загружены, и физические поля адреса заголовочного файла секции ELF тогда обращаются секциям, относящихся к i386 документации ELF для деталей относительно того, как читать секции заголовочного файла. Обратите внимание, что shdr_num может быть 0, не указывая никакие символы, даже если бит 5 в слове флагов установлен. Если бит 6 в слове флагов установлен, то mmap_* поля допустимы, и указывают адрес и длину буфера, содержащего карту памяти машины, обеспеченной BIOS'ом. Mmap_addr - адрес, а mmap_length - полный размер буфера. Буфер состоит из одного, или большего количества следующих size/структурных пар (размер действительно используется для пропуска к следующей паре): 

 4 Size 0 base_addr_low 

 4 Base_addr_high 8 Length_low 12 Length_high 16 Typewhere size - размер связанной структуры в байтах, которые могут быть больше чем минимум 20 байтов. Base_addr_low - низкие 32 бита начального адреса, а base_addr_high - верхние 32 бита для общего количества 64-разрядного начального адреса. Length_low низкие 32 бита из размера области памяти в байтах, а length_high - верхние 32 бита для общего количества 64 битов. Type - разнообразие представленного адресного интервала, где значение 1 указывает доступную оперативную память, а все другие значения в настоящее время обозначают резервную область. 

 Будет выведено значение оперативной памяти, необходимое для нормальной работы.

 Если бит 7 во флагах установлен, то drives_* поля допустимы, и указывают адрес физического адреса первой структуры диска и размера структур диска. Drives_addr - адрес, и drives_length - полный размер структур диска. Обратите внимание, что drives_length может быть нулевой. Каждая структура диска отформатирована следующим образом: 

 0 Size 4 Drive_number 5 Drive_mode 6 Drive_cylinders 8 Drive_heads 9 Drive_sectors 10-xx Drive_ports Поле size определяет размер этой структуры. Размер меняется в зависимости от числа портов. Обратите внимание, что размер не может быть равен (10 + 2 * число портов), из-за линеаризации. 

 Поле Drive_number содержит номер диска BIOS. Поле Drive_mode представляет способ доступа, используемый загрузчиком ядра. В настоящее время определены следующие способы: 

0 CHS mode (традиционный cylinder/head/sector способ адресации) 1 LBA mode (логический способ адресации блока) Эти три поля, drive_cylinders, drive_heads и drive_sectors указывают геометрию диска, обнаруженного BIOS'ом. Drive_cylinders содержит число цилиндров. Drive_heads содержит число глав. Drive_sectors содержит число секторов в дорожке.

 Поле Drive_ports содержит матрицу портов ввода - вывода, используемых для диска в коде BIOS. Массив состоит из нуля или больших нуля целых чисел с двумя байтами, и заканчиваевшихся нулем. Обратите внимание, что массив может содержать любое число портов ввода - вывода, которые не связаны с диском фактически (типа портов DMA контроллеров). 

 Если бит 8 во флагах установлен, то поле config_table допустимо и указывает адрес таблицы конфигурации ПЗУ, возвращенной на вызовы BIOS. Если вызовы BIOS остались без ответа, то размер таблицы должен быть нулевой. 

 Если бит 9 во флагах установлен, то поле boot_loader_name допустимо, и содержит физический адрес имени загрузчика ядра, при загрузке ядра. Имя - законченная нулем строка в C-стиле. 

 Если бит 10 во флагах установлен, то поле apm_table допустимо, и содержит физический адрес определенной таблицы APM как указано ниже : 

 0 version 2 Cseg 4 Offset 8 Cseg_16 10 Dseg 12 Flags 14 Cseg_len 16 Cseg_16_len 18 Dseg_len Поля version, cseg, offset, cseg_16, dseg, флаги, cseg_len, cseg_16_len, dseg_len указывает номер версии, защищенный режим 32-разрядный сегмент кода, смещение точки входа, защищенный режим 16-разрядный сегмент кода, защищенный режим 16-разрядный сегмент данных, флаги, продолжительность защищенного режима 32-разрядного сегмента кода, продолжительность защищенного режима 16-разрядного сегмента кода, и продолжительность защищенного режима 16-разрядного сегмента данных, соответственно. Только поле offset - 4 байта, а остальные - 2 байта. См. расширенное управление электропитанием (APM) спецификации интерфейса BIOS для получения дополнительной информации. 

 Бит 11 во флагах должен быть нулевым. 

<h2><a name="для_разработчиков_драйвера_устанавливаемой_файловой_системы" id="для_разработчиков_драйвера_устанавливаемой_файловой_системы">Для разработчиков драйвера устанавливаемой файловой системы</a></h2> 

Устанавливаемые драйверы файловой системы (IFS) описаны в документе по IFS (еще не издан). 

<h2><a name="устройство_загрузчика_ядра" id="устройство_загрузчика_ядра">Устройство загрузчика ядра</a></h2> 

 Загрузчик ядра пока ещё сырой 16/32-разрядный двоичный код (подобно <acronym title="Microsoft">MS</acronym>/PC-ДОСУ COM файлам, но запускается не от 100h, а от 0h).

 <ul> <li class="level1"> Прежде всего, загрузчик ядра хранит всю информацию от регистров центрального процессора во внутренних структуры

</li> <li class="level1"> Выдается информация относительно памяти </li> <li class="level1"> После этого выводится информация на дисплей </li> <li class="level1"> Загружается ядро </li> <li class="level1"> Переключается в защищенный режиму </li>

<li class="level1"> И выполняется мультизагрузка ядра </li> </ul>

Пока всё! Не так-то это просто! <img src="/web/20100304104902im_/http://www.osfree.org/wiki/lib/images/smileys/icon_wink.gif" class="middle" alt="" />

From:
http://osfree.org/doku/ - **osFree wiki**

Permanent link:
**http://osfree.org/doku/doku.php?id=ru:docs:boot:freeldr&rev=1363011902**

Last update: **2013/03/11 14:25**