Министерство образования и науки Российской Федерации
Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение высшего профессионального образования

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет
Специальность 010503 «Математическое обеспечение и администрирование информационных систем»
Кафедра системного программирования

ДИПЛОМНАЯ РАБОТА

Flash файловая система OCPB Embox

Батюков А.М.
группа 545

Руководитель …… асп. Бондарев А.В.

Рецензент …… Венгеров В.В.

«Допущен к защите» …… д.ф.-м. н., проф.
Заведующий кафедрой А.Н.Терехов

Санкт-Петербург
2011

Saint-Petersburg State University

Mathematics and Mechanics Faculty

Software Engineering Department

Flash filesystem for Embox RTOS

Graduate paper by
Alexander Batyukov
545 group

Scientific advisor	……	Bondarev A.V.
Reviewer	……	Vengerov V.V.
"Approved by" Head of Department	……	Professor A.N. Terekhov

Saint-Petersburg

2011

Оглавление

Глава 1. Введение

1.1 Предметная область

Разработка встраиваемых систем (Embedded Systems) – особая область системного программирования, направленная на разработку программно-аппаратных комплексов разного уровня. Она активно развивается в последнее время в связи с ростом применения микрокомпьютеров и микроконтроллеров в различных устройствах.

Одной из ключевых особенностей этой области является направленность на эффективное решение конкретных задач, зачастую реального времени, в условиях ограниченности аппаратных ресурсов.

Эффективного решения задачи разработки встраиваемых систем можно добиться лишь при активном сотрудничестве и взаимодействии как программистов, так и инженеров, ищущих пути решения на стыке программирования и проектирования аппаратуры.

Таким образом, с точки зрения системного программиста, разработка встраиваемых систем сегодня представляет собой совершенно особую область, в которой зачастую неприменимы классические подходы решения тех или иных задач.

1.2 Проект Embox

В рамках проекта Embox[1] ведется разработка операционной системы реального времени (ОСРВ) для встраиваемых систем. Ключевой особенностью

[1] ОСРВ Embox, режим доступа: http://code.google.com/p/embox

этой системы является модульность и конфигурируемость, что позволяет применять ее для широкого класса встраиваемых устройств. Кроме того, исторически сложилось, что одним из направлений развития системы является создание на ее основе инструментального средства для разработки встраиваемых систем.

ОСРВ Embox второй год разрабатывается на кафедре Системного программирования Математико-механического факультета СПбГУ. В ее основу положены наработки кафедры и специалистов ЗАО «Ланит-Терком».

1.3 Flash-память

Системное программное обеспечение (СПО), созданное программистом, загружается на аппаратуру и хранится там постоянно. Оно представляет собой, вообще говоря, код системного загрузчика и (возможно) операционной системы, в которой реализованы управляющие алгоритмы. Кроме того, оно должно оставлять возможность конфигурирования под конкретный вид используемой аппаратуры.

Код загрузчика, операционная система (ОС) и конфигурационные файлы хранятся прямо на аппаратуре в энергонезависимой памяти. Стандартом де-факто во встраиваемых системах стало использование flash-памяти. Эта память обладает определенными достоинствами и недостатками, которые перечислены ниже.

Достоинства:

- энергонезависимость;
- механическая стойкость;
- высокая скорость чтения;
- малый физический размер.

Недостатки:

- относительная дороговизна;
- недолговечность;
- невысокая скорость перезаписи.

Существуют два основных типа flash-памяти в зависимости от используемой технологии – NOR (на основе элемента ИЛИ-НЕ) и NAND (на основе И-НЕ). Для NOR-памяти характерна довольно высокая скорость чтения. Более современная NAND-память поддерживает большие ёмкости и гораздо более высокую скорость записи и стирания, но интерфейс ввода/вывода NAND-памяти намного сложнее. Кроме того, в NAND-памяти битовые ошибки появляются на порядок чаще, чем в NOR-памяти, что делает необходимым использование корректирующих ошибки кодов.

Устройства flash-памяти, как правило, имеют внутреннее деление на разделы, доступ к которым может производиться параллельно (например, стирание одного раздела одновременно с чтением из другого). Разделы в свою очередь делятся на блоки (или ячейки, erasable block) размером обычно 64 или 128 КБ. В отличие от других носителей, таких как, например, RAM-диски, flash-носители имеют некоторые особенности. Допускается изменить отдельный бит flash-памяти с единицы на ноль. Если необходимо изменить бит с нуля на единицу, то стирается весь блок – для перевода всех битов в одно состояние. Из этого следует, что остальные данные блока требуется сохранить в другом месте. Обычно NOR-память принимает данные по одному байту, в то время как в NAND-память возможно производить запись постранично.

Оба типа flash-памяти поддерживают специальную команду стирания (erase) всего блока. Кроме того, NOR-память требует, чтобы перед стиранием все биты блока были явно установлены в ноль. Как правило, операция стирания

занимает значительное количество времени: для NOR-памяти порядка нескольких секунд, а для NAND-памяти – миллисекунд. Ключевой характеристикой flash-памяти является допустимое количество операций стирания. Каждый блок NOR-памяти возможно стирать до 100000 раз, блок NAND-памяти – до миллиона [7].

NAND и NOR-архитектуры сейчас существуют параллельно и не конкурируют друг с другом, поскольку находят применение в разных областях хранения данных. В современных встроенных системах для хранения СПО часто используют flash-память типа NOR, объемом от нескольких килобайт до гигабайт.

1.4 Файловые системы

В данной работе рассматриваются вопросы разработки файловых систем. Область проектирования и реализации файловых систем стала уже классической в системном программировании.

В основе понятия файловой системы лежит концепция файла. Первоначально понятие «файл» было довольно узким, под ним понималась именованная совокупность данных. На сегодняшний день понятие «файла» в общем случае можно сформулировать следующим образом: файл – это некоторая абстракция, обладающая некоторыми характеристиками и предоставляющая доступ к определенным ресурсам [6].

Например, такими ресурсами могут быть:

- области данных (необязательно на диске);
- устройства (как физические, так и виртуальные);
- потоки данных;
- сетевые ресурсы;

- объекты операционной системы.

Файлы управляются посредством файловой системы, которая, как правило, является частью операционной системы. Файловой системой называется некоторая система организации данных и метаданных на устройстве хранения. Это достаточно общее определение, которое сложилось за всю историю развития и понимания концепции хранения данных. Оно отражает только основные принципы, не давая указаний к способам их реализации.

Следует отметить, что развитие файловых систем персональных компьютеров определялось, прежде всего, двумя факторами:

- появлением новых стандартов на носители информации;
- ростом требований к характеристикам файловой системы со стороны прикладных программ.

В попытках создания универсальных файловых систем, пригодных для различных носителей информации и удовлетворяющих всем выдвигаемым к ним требованиям, было предложено множество оригинальных концепций, которые достаточно успешно решают эту задачу. Это естественным образом привело к возрастанию сложности подобных систем и их требований к аппаратным ресурсам, что в свою очередь сделало неудобным или даже невозможным применение их в разработке встраиваемых систем.

В последнее время все чаще делаются попытки создания удобной файловой системы для применения во встраиваемых системах различного уровня. За последнее десятилетие появились и успешно развиваются в своей области файловые системы JFFS/JFFS2, YAFFS, UFFS, coffee и многие другие. Каждая из них имеет свои собственные достоинства и недостатки и применяется в своей области. Исследования в области файловых систем для

встраиваемых систем можно вести только основываясь на опыте создания подобных систем.

1.5 Постановка задачи

В рамках развития проекта Embox появилась потребность наличия в системе абстрагированного интерфейса доступа к ресурсам, что привело к необходимости разработки файловой системы.

В первую очередь необходимо выделить ключевые особенности, которыми должна обладать реализуемая система. Далее нужно проанализировать существующие файловые системы на предмет возможности использования их наработок в проекте. Конечным итогом должна стать реализация файловой системы в соответствии с концепциями проекта Embox.

1.6 Требования к системе

Проектируемая система должна обладать следующими ключевыми особенностями:

- универсальный интерфейс (по возможности POSIX[2]-совместимый) доступа к файлам;
- ориентированность на flash-память;
- возможность работы с ram-памятью;
- нетребовательность к ресурсам и достаточная скорость работы;
- удобство для применения на этапе разработки встраиваемых систем.

[2] POSIX, Portable Operating System Interface for Unix, режим доступа: http://standards.ieee.org/develop/wg/POSIX.html

Необходимо, чтобы разрабатываемая система была модульной по своей структуре. Кроме того, должна быть возможность адаптировать ее к конкретной аппаратуре и решаемым в каждом конкретном случае задачам.

Глава 2. Обзор концепций файловых систем

2.1 Интерфейс разделов жесткого диска

Раздел (partition) – часть долговременной памяти жесткого диска, выделенная для удобства работы и состоящая из смежных блоков. На других носителях информации выделение разделов не предусмотрено и почти не практикуется. Интерфейс разделов представляет физический диск как набор логических, позволяя использовать на каждом из них свою собственную файловую систему. Таким образом, он представляет собой «прослойку» между физическим уровнем и уровнем файловых систем ([8], [9]).

Достоинства:

- на одном жестком диске можно хранить информацию о разных файловых системах;
- можно отделить информацию пользователя от информации ОС;
- позволяет установить несколько ОС на один физический диск.

Информация о размещении разделов на жестком диске хранится в таблице разделов, которая является частью главной загрузочной записи (Master Boot Record, MBR), всегда располагающейся в первом физическом секторе жесткого диска. Каждый раздел может быть либо первичным (основным), либо дополнительным. В первом секторе каждого основного раздела находится загрузочный сектор (Boot Record), отвечающий за загрузку операционной системы с этого раздела. Информация о том, какой из основных разделов будет использован для загрузки операционной системы, тоже записана в главной загрузочной записи [8].

Интерфейс разделов жесткого диска может быть перенесен на flash-накопители, но не предназначен изначально для работы с ними, что создает ряд трудностей, таких как быстрый износ flash-ячеек. Кроме того, он в достаточной степени низкоуровневый.

2.2 Linux: виртуальный коммутатор файловых систем

Виртуальный коммутатор файловых систем ОС Linux (Virtual filesystem Switch, VFS) – высокоуровневая абстракция, отделяющая POSIX API от подробностей работы конкретной файловой системы. VFS предоставляет общую файловую модель, которую наследуют нижележащие файловые системы (они должны реализовать действия для различных функций POSIX API). Дальнейшее абстрагирование, за пределами VFS, скрывает находящееся ниже физическое устройство (которое может являться диском, разделом диска, сетевым модулем хранения, памятью или любым другим носителем, способным хранить информацию, даже временно). В дополнение к сокрытию деталей файловых операций от лежащих ниже файловых систем VFS «привязывает» нижележащие блочные устройства к имеющимся файловым системам [5].

Linux не был первой операционной системой, использующей виртуальный уровень для поддержки общей файловой модели. Примерами ранних реализаций VFS являются Sun VFS (в SunOS версии 2.0, приблизительно 1985 год) и Installable File System от IBM и Microsoft для IBM OS/2. Эти подходы к виртуализации уровня файловой системы проложили дорогу для Linux VFS.

Виртуальный коммутатор файловых систем удобен в использовании и является необходимой частью современной файловой системы.

2.3 Файловая система FAT32

Файловая система FAT (File Allocation Table, FAT) – классическая архитектура файловой системы. Иерархическая по своей структуре. Смежные секторы диска объединяются в единицы, называемые кластерами. Количество секторов в кластере может быть равно 1 или степени двойки. Для хранения данных файла отводится целое число кластеров.

Пространство тома FAT32 логически разделено на три смежные области:

- зарезервированная область. Содержит служебные структуры, которые принадлежат загрузочной записи раздела (Partition Boot Record, PBR) и используются при инициализации тома;
- область таблицы FAT, содержащая массив индексных указателей («ячеек»), соответствующих кластерам области данных. Обычно на диске представлено две копии таблицы FAT в целях надежности;
- область данных, где записано собственно содержимое файлов и метаданные.

Запись и чтение происходят кластерами, при удалении файла первый знак имени заменяется специальным кодом, и цепочка кластеров файла в таблице размещения обнуляется [8].

Достоинства:

- повсеместная поддержка;
- сравнительно быстрая работа с flash-памятью [12].

Файловая система семейства FAT может быть использована для flash-накопителей, но приводит к сильному износу ячеек flash-памяти, отведенных под таблицу размещения файлов. Кроме того, некоторые другие области (например, место размещения корневого каталога) также могут часто перезаписываться. Все это ограничивает область применения файловых систем

семейства FAT только теми flash-накопителями, у которых контроллер аппаратно поддерживает алгоритм выравнивания износа (wear leveling) блоков.

2.4 Linux: файловые системы семейства ext

Ext2 (Extended File System version 2, Ext2) – файловая система ядра ОС Linux, подключаются к виртуальному коммутатору файловых систем. Ext2 – файловая система с сетевой организацией, что позволяет файлам входить сразу в несколько каталогов. Особенностью системы является то, что атрибуты файлов хранятся не в каталогах, как это сделано в ряде простых файловых систем, а в специальных таблицах. В результате каталог имеет очень простую структуру, состоящую всего из двух частей: номера индексного дескриптора и имени файла. Запись и чтение происходят блоками, информация о блоках хранится в индексной таблице. Ведется резервирование важных частей файловой системы [4].

Достоинства:

- очень высокая скорость работы для жестких дисков;
- надежность хранения данных;
- открытость.

Основное отличие файловой системы ext3 от описанной выше ext2 в том, что она журналируема. Журналируемой также является и более современная файловая система ext4. Ее отличает наличие механизма пространственной (extent) записи файлов (новая информация добавляется в конец заранее выделенной по соседству области файла) и некоторые другие оптимизации алгоритмов выделения памяти, уменьшающие фрагментацию и повышающие производительность системы. [4]

Журналируемость файловых систем ext3/ext4 – важное свойство, позволяющее восстанавливать файловую систему при сбоях, но влекущее за собой избыточное количество операций записи во flash-ячейки. Количество записанной информации возрастает от 4% – 12% для операций чтения/записи до 55% для операций стирания/перезаписи данных. Современные носители информации достаточно вместительны, поэтому этим обычно пренебрегают и используют эти файловые системы даже в некоторых встраиваемых системах. Но такое решение подходит не для всех встраиваемых систем [3].

Кроме того, во избежание быстрого износа ячеек flash-памяти файловые системы семейства ext необходимо монтировать с опцией noatime, которая запрещает запись отметок о времени доступа к файлам. Это следствие того, что изначально ext2 не была ориентирована на применение на flash-носителях.

2.5 Linux: файловые системы JFFS/JFFS2

JFFS (Journaling Flash File System, JFFS) – современная журналируемая flash-ориентированная (NOR) файловая система, подключаемая к виртуальному коммутатору файловых систем.

JFFS обеспечивает равномерность износа (wear leveling) ячеек flash-памяти за счет реализации циклического алгоритма перезаписи. Ячейки памяти организованы в список. При изменении файловой системы запись ведется в конец этого списка. В начале каждой ячейки записываются метаданные, за которыми следуют полезные данные. Ячейки связаны между собой указателями, содержащимися в метаданных. Процесс изменения данных происходит следующим образом: ячейка, содержащая старую версию данных помечается устаревшей, измененные данные копируются в новую свободную ячейку, после чего переставляются указатели. Когда свободных ячеек памяти

становится мало, «сборщик мусора» освобождает (erase) устаревшие ячейки, копируя неустаревшие из «головы» в «хвост» списка ([2], [7]).

JFFS2 отличается от JFFS поддержкой NAND flash-памяти, наличием сжатия данных и несколько модифицированным алгоритмом «сборки мусора». В JFFS2 все ячейки организованы в три списка: в первом содержатся ячейки, содержащие только неустаревшие элементы файловой системы информацию, во втором – те из них, которые содержат хотя бы один устаревший элемент, в третьем – свободные блоки. Алгоритм «сборки мусора» в 99% случаев перемещает неустаревшую информацию из ячейки во втором списке в третий, а в 1% случаев просто перемещает информацию из ячейки в первом списке в третий ([2], [7]).

Достоинства:

- flash-ориентированные;
- журналируемые.

Файловые системы JFFS/JFFS2 часто используются для загрузки различных конфигурационных файлов в ОС Linux, когда эта ОС используется во встраиваемых системах не сильно ограниченных по ресурсам. Пожалуй, являются лучшим решением для традиционного использования flash-накопителей. По некоторым исследованиям, эти файловые системы слишком требовательны к объему оперативной памяти вследствие необходимости хранить в памяти информацию о размещении ячеек на flash-устройстве. Кроме того, процесс монтирования этих файловых систем может быть очень длительным из-за необходимости сканирования всего объема flash-памяти для определения распределения по ним информации. Они не подходят для использования в сильно ограниченных по ресурсам встраиваемых системах [13].

2.6 Contiki: файловая система coffee

Coffee – файловая система проекта Contiki, в рамках которого идет разработка ОС для встраиваемых систем. Flash-ориентированная, поддерживает алгоритм циклической перезаписи ячеек [10].

Достоинства:

- flash-ориентированная;
- интуитивно понятный интерфейс;
- не требовательна к ресурсам системы.

Основной особенностью данной файловой системы является то, что для описания каждого файла используется фиксированное (и небольшое) количество оперативной памяти системы. При создании файла в его начало помещается структура, содержащая в себе ссылки на занимаемые файлом ячейки памяти. В момент модификации файла создается так называемый микролог, ссылка на который также помещается в структуру, описывающую файл. Микролог хранит модификации файла как последовательный набор записей о модификации данных. Когда микролог заполняется определенным количеством записей, путем слияния создается новый файл с новой описывающей его структурой, а старые файлы удаляются из системы.

Описанные принципы позволяет использовать файловую систему coffee во встраиваемых устройствах, сильно ограниченных по ресурсам. Файловая система coffee реализует необходимый минимум функциональности работы с файлами, поддерживает POSIX-интерфейс. Архитектура системы максимально упрощена в угоду минимизации использования ресурсов.

2.7 Файловая система UFFS

Файловая система UFFS (An ultra lowcost flash file system, UFFS) – результат разработки ультра-лёгкой flash-ориентированной файловой системы для встраиваемых систем.

Достоинства:

- экстремальная минималистичность;
- быстрое монтирование ФС;
- статическое распределение всей памяти при загрузке;
- может работать без ОС.

Все flash-блоки организованы в древоподобную структуру: каждому из них присваивается метка, состоящая из двух частей. Первая часть – указатель на уникальный номер родительского блока, вторая часть – собственный уникальный номер. Дерево строится и хешируется в оперативной памяти в момент монтирования файловой системы. Используется циклический алгоритм перезаписи и алгоритм уплотнения информации в блоках. Поддерживается избыточное кодирование информации в целях увеличения надежности работы системы.

Перечисленные особенности UFFS направлены на максимально быструю работу с файлами, что, по заявлениям разработчиков, позволяет использовать данную файловую систему даже в задачах, требующих «мягкого» реального времени.

Файловая система UFFS в настоящее время поддерживает работу только с NAND типом flash-памяти. Кроме того представляет специфичный интерфейс «страниц» вместо привычного прикладному программисту интерфейса файлов [13].

2.8 Файловые системы Cramfs и SquashFS

Иногда во встраиваемых системах нет нужды модифицировать данные файловой системы и достаточно только чтения. Наиболее популярные файловые системы, предоставляющие доступ к данным в режиме «только для чтения» (read only) – это cramfs и SquashFS.

Cramfs хранит данные в сжатом виде и хорошо подходит для flash-носителей. Основные ее черты – простота и компактность, поэтому она незаменима в небольших маломощных устройствах. В cramfs метаданные хранятся в несжатом виде, а сами данные организованы в виде страниц, каждая их которых сжимается по алгоритму zlib, что дает возможность произвольного доступа к страницам.

SquashFS – еще одна сжатая файловая система только для чтения. Часто ее можно встретить на многих LiveCD-дистрибутивах Linux. В SquashFS используется как сжатие zlib, так и более быстрый и эффективный алгоритм LZMA.

Подобные файловые системы удобны для применения во встраиваемых устройствах на этапе их эксплуатации, когда flash-память используется только как хранилище стартового кода системы, но они неприменимы на этапе разработки тех систем, в которых во время тестирования требуется частая перезапись как кода системы, так и конфигурационных файлов.

Кроме рассмотренных существует еще большое количество как файловых систем общего назначения (таких как NTFS, btrfs, raiserfs/raiser4, ISO 9660), так и flash-ориентированных файловых систем (например, FFS/FFS2, ETFS, ExtremeFFS, YAFFS). Они ничем не хуже, а в чем-то даже лучше рассмотренных. Выбирая, какие из них описать, я руководствовался желаниями

описать базовые принципы и дать как можно более широкий обзор современных файловых систем. Кроме того предпочтение отводилось описанию файловых систем, которые оказали влияние на написание файловой системы ОСРВ Embox ([7] - [9]).

В результате обзора существующих концепций файловых систем мною был сделан вывод о наличии разрыва между файловыми системами общего назначения и файловыми системами, предназначенными прежде всего для встраиваемых систем. Первые становятся все более универсальными за счет многоуровневости и модульности, но жертвуют при этом скоростью работы и становятся все более требовательными к ресурсам. Вторые нацелены на быстрое решение только узкого класса задач, но, как правило, одноуровневые и немодульные, кроме того, зачастую непривычны в использовании.

Можно попытаться объединить положительные стороны каждого из классов файловых систем и с учетом специфики предметной области создать собственную модульную файловую систему для встраиваемых систем.

Глава 3. Теоретическая часть

3.1 Общая структура файловой системы

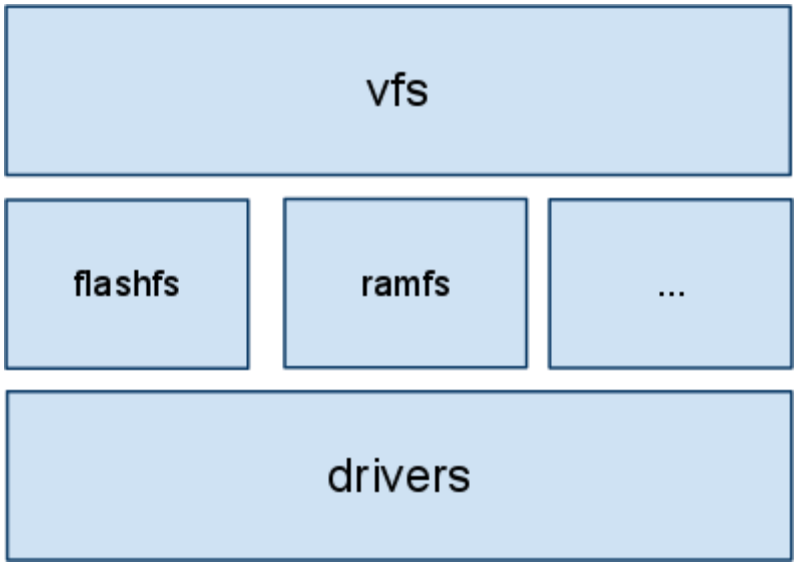

Рис.3.1. Общая структура файловой системы

Организация файловой системы – трёхуровневая. На верхнем уровне находится виртуальная файловая система (Virtual File System, VFS), в которой реализован общий интерфейс работы с файлами. Уровнем ниже располагаются драйверы файловых систем, реализующие функции интерфейса верхнего уровня в зависимости от особенности той или иной конкретной файловой системы. На самом нижнем слое находится реализация аппаратно-зависимых драйверов.

Такая организация файловой системы с одной стороны предоставляет универсальный интерфейс доступа к файлам, находящимся на различных носителях, а с другой стороны очень конфигурируема, позволяет использовать только необходимые модули и легко добавлять новые.

3.2 Модульность

Каждый элемент файловой системы (vfs, драйвера файловых систем, драйвера устройств) оформлен в отдельный независимо подключаемый модуль. Это позволяет тонко настраивать систему под конкретную аппаратуру, выбирая только необходимые для функционирования конкретного устройства модули. Все модули, реализованные в системе, могут покрывать достаточно обширную функциональность, но в каждом конкретном случае используется только необходимый срез из них.

3.3 Алгоритмы уменьшения износа flash-ячеек

Для уменьшения износа ячеек flash-памяти используется алгоритм трансляции реальных «физических» адресов в доступные через стандартизованный интерфейс «логические» адреса. С одной стороны, такое преобразование позволяет работать с накопителем в рамках распространенных стандартов без применения специализированного ПО. С другой стороны, оно призвано обеспечить прозрачный для пользователя дефект-менеджмент и выравнивание «износа» ячеек памяти, обладающих ограниченным ресурсом: при необходимости перезаписи ячейки информация из нее переносится в свободную ячейку памяти с наименьшим износом. Это возможно за счет информации, содержащейся в таблице трансляций, которая описана в пункте 4.4.1.

3.4 Перепрограммирование flash-памяти

Вообще говоря, код загрузчика и операционной системы во встраиваемых системах может исполняться как из оперативной, так и непосредственно из

flash-памяти. Мне на практике приходилось работать с системой, код загрузчика и некоторых специфичных функций которой выполнялся напрямую из flash-памяти (в первую очередь исходя из соображений надежности системы). В этом случае аппаратные ограничения не позволяют одновременно с исполнением кода производить операции записи во flash-память, ограничивая, таким образом, ее использование режимом только для чтения. Причём на этапе использования некоторых встраиваемых систем такого режима зачастую хватает, ведь в этом случае во flash-памяти, как правило, просто хранятся данные, необходимые для загрузки и конфигурирования системы, а вся работа происходит в оперативной памяти. Но на этапе разработки встраиваемых систем достаточно часто возникает особая ситуация загрузки нового образа системы для тестирования. Эта ситуация характеризуется тем, что в этот момент целью является обновление образа самой системы, и она некоторое время, требуемое для обновления, может не функционировать. При использовании файловых систем, предоставляющих режим только для чтения, обновление системы приходится производить с помощью сторонних устройств, называемых перепрограмматорами. Эту неудобно, в первую очередь вследствие достаточно высокой цены этих устройств.

Чтобы иметь возможность перепрограммирования системы без использования дополнительных устройств, было решено специфичные функции драйверов, отвечающие непосредственно за запись во flash-память, поместить в отдельную перемещаемую секцию. При загрузке системы эта секция копируется в оперативную память. Это позволяет в момент перепрограммирования системы исполнять код из оперативной памяти. Например, появляется возможность загрузить код образа в оперативную память через ethernet-кабель по tftp протоколу и после этого записать его во flash-память.

3.5 Realtime возможности

Работа системы в режиме реального времени предполагает предсказуемость реакции системы на внешние события. Такая предсказуемость складывается из предсказуемости времени работы отдельных подсистем, в частности, файловой системы. Реализация realtime файловой системы может быть куда менее эффективной, чем «традиционной», и содержать на порядок больше ограничений. Но организация информации на носителе и структуры данных в памяти должны быть такими, чтобы операции доступа к файловой системе осуществлялись детерминированно.

Общая задача создания flash-ориентированной файловой системы реального времени, несмотря на все особенности, может быть решена, но это не являлось целью данной работы.

Реализованная файловая система не является системой реального времени. В общем случае, структуры хранения vfs не обладают свойством детерминированности времени обращения к данным. Кроме того, реализация драйвера файловой системы flashfs также не обладает данной особенностью по отношению к операциям записи. Тем не менее, в системе реализованы некоторые возможности, позволяющие с некоторыми ограничениями получить доступ к файлам с фиксированным временем операций (чтения и записи для ram, чтения для flash). Это возможно за счет использования механизма, описанного в пункте 4.5.

Глава 4. Практическая часть

4.1 Инструментальные средства

Разработка проекта Embox ведется преимущественно на языке C в рамках среды разработки Eclipse[3]. Для конфигурирования и сборки проекта используется разработанная в рамках проекта оригинальная система сборки, в основе которой лежит язык make. В качестве компилятора используется gcc.

Так как файловая система является подсистемой Embox, ее реализация также использует данные инструментальные средства. Кроме того в некоторой мере сам проект Embox является инструментальным средством для разработки файловой системы, так как является платформой, на основе которой стало возможно создание такой системы.

В качестве системы контроля версий используется Subversion (SVN)[4], в качестве сервера для хранения проекта – хостинг проектов с открытым исходным кодом googlecode[5], который предоставляет собой платформу для распределенной разработки, включающую в себя, кроме репозитария, механизмы распределения задач (issue-трекер) и написания документации (wiki-редактор).

[3] IDE Eclipse, режим доступа: http://eclipse.org
[4] Subversion, режим доступа: http://subversion.tigris.org/
[5] Google code, режим доступа: http://code.google.com

4.2 Виртуальная (логическая) файловая система

Каждый файл в системе представляет собой узел (node) виртуальной файловой системы. Все узлы организованы в иерархию, общая структура которой – дерево, в корне которого лежит узел «/». Узел описывается специальной структурой:

```
typedef struct node {
        const char              name[CONFIG_MAX_LENGTH_FILE_NAME];
        void                    *file_info;
        file_system_driver_t    *fs_type;
        struct list_head        neighbors;
        struct list_head        leaves;
} node_t;
```

Элементы структуры имеют следующее предназначение:

- *name* - имя узла в системе;
- *file_info* - некая внутренняя информация о файле;
- *fs_type* - указатель на драйвер файловой системы, обслуживающей этот узел;
- *neighbors* - структура, содержащая информацию о «соседях» того же уровня;
- *leaves* - структура, содержащая информацию о «детях» узла**.**

4.3 Пример организации файловой системы

Драйвер файловой системы описывается структурой:

```
typedef struct file_system_driver {
        const char                      *name;
        const file_operations_t         *file_op;
        const fsop_desc_t               *fsop;
} file_system_driver_t;
```

Элементы структуры:

- *name* - имя драйвера файловой системы;
- *file_op* - указатель на структуру, описывающую операции, которые могут быть произведены с файлом;
- *fsop* - указатель на структуру, описывающую операции, которые могут быть произведены с файловой системой.

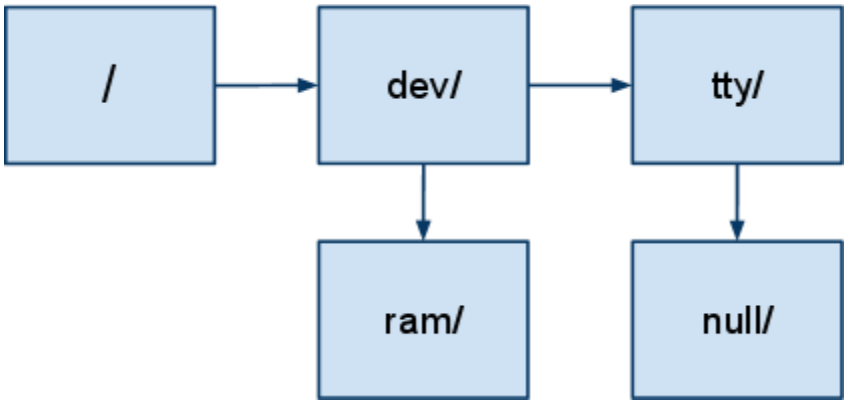

Рис. 4.1. Пример организации файловой системы

Для того чтобы добавить поддержку нового драйвера файловой системы, необходимо реализовать все необходимые функции, после чего зарегистрировать драйвер в виртуальной файловой системе с помощью макроса DECLARE_FILE_SYSTEM_DRIVER, который добавит описание драйвера в специальную секцию. Таким образом, мы статически можем задать все необходимые параметры драйвера файловой системы.

4.4 Подключаемые драйвера файловых систем

В момент создания узла виртуальной файловой системы в описывающую его структуру вносится информация о том, драйвер какой файловой системы используется для операций с этим узлом. В файловой системе проекта Embox

реализованы два драйвера файловых систем: flashfs и ramfs. Каждый из них реализует необходимые функции интерфейса виртуальной файловой системы vfs.

4.4.1 Особенности реализации flashfs

Вся информация о файловой системе расположена в нескольких таблицах, хранимых в зарезервированных областях flash-памяти.

Таблица файлов

Таблица файлов располагается в первых двух ячейках flash-памяти. Одна из ячеек в каждый момент времени зарезервирована и не содержит полезной информации, другая – рабочая. Рабочая ячейка организована по принципу микролога, она постепенно заполняется элементами следующей структуры:

```
typedef struct _file_entry {
        uint32_t        file_id;
        uint32_t        start_ block;
        uint32_t        parent_id;
} file_entry_t;
```

Каждый элемент при этом описывает соответствие между идентификатором файла и логическим номером ячейки flash-памяти (стартовой ячейки для файла), в которой содержится информация об этом файле. Кроме того в структуре содержится ссылка на идентификатор родительского файла. В рабочей ячейке может быть одновременно несколько структур, описывающих файл с одним и тем же идентификатором, корректной считается структура, находящаяся дальше всех от начала ячейки, остальные считаются устаревшими. При добавлении нового файла или изменении стартовой ячейки текущего файла информация об этом дописывается в конец микролога до тех пор, пока микролог не заполнит всю рабочую ячейку. В этот момент происходит копирование всех

неустаревших записей из рабочей ячейки в зарезервированную и очищение рабочей ячейки. После этого ячейки меняются ролями.

Использование такой таблицы накладывает ограничение на число возможных файлов в системе. Исходя из размера ячейки 64КБ (что соответствует размеру конфигурационных ячеек используемой flash-памяти) и размера одной записи в 12Б, максимально возможное число файлов в системе оценивается ~5000.

Стартовый блок файла

Стартовый блок каждого файла также организован по принципу микролога. Он состоит из записей следующего типа:

```
typedef struct _file_description {
        uint32_t        file_id;
        uint32_t        prev_ block;
        uint32_t        next_block;
        uint32_t        old_me;
} file_description_t;
```

Каждая запись описывает одну из логических ячеек, принадлежащих файлу. В микрологе может одновременно находиться несколько записей, описывающих один и тот же блок. При этом корректной считается та из них, что находится дальше от начала стартовой ячейки. Все корректные записи в любой момент времени организованы в виде двунаправленного списка. При добавлении в файл новых логических ячеек или изменении существующих информация об этом дописывается в конец микролога, при этом каждая измененная ячейка содержит ссылку на устаревшую версию самой себя для отслеживания устаревших ячеек. Когда микролог заполнит всю стартовую ячейку, все неустаревшие записи из него копируются в новую ячейку, которая становится стартовой для файла (и информация об этом записывается в таблицу файлов).

Использование стартового блока файла таким образом накладывает ограничение на максимальный размер файла в системе. Исходя из размера ячейки в 256КБ (что соответствует размеру стандартной ячейки используемой flash-памяти) и размера одной записи в 16 байт, максимальный размер файла составляет 2ГБ. Кроме того, использование стартового блока в текущей реализации ограничивает минимальный размер файла на flash-диске размером одной ячейки.

Таблица трансляций

Логический номер ячейки – это некоторое число, которым оперируют все структуры, связанные с файлами. Каждая логический номер ячейки в каждый момент времени сопоставлен одной физической ячейке flash-памяти. В разные моменты времени одному и тому же логическому номеру могут соответствовать разные физические ячейки. Информация о соответствии логических номеров физическим ячейкам содержится в таблице трансляций. Эта таблица расположена в двух ячейках, находящихся непосредственно за ячейками, содержащими таблицу файлов, и организована очень похожим на таблицу файлов способом. Одна из ячеек в каждый момент времени зарезервирована, а вторая (рабочая) устроена по принципу микролога. Микролог заполняется записями вида:

```
typedef struct _block_entry {
        uint32_t        logical_address;
        uint32_t        occupied;
        uint32_t        physical_address;
        uint32_t        age;
} block_entry_t;
```

Каждая запись содержит информацию о соответствии логического номера физической ячейке, размер занятого пространства внутри ячейки и возраст

соответствующей физической ячейки. В каждый момент времени в рабочей ячейке может быть несколько записей об одном логическом номере, корректной считается находящаяся дальше от старта ячейки. При добавлении или изменении информации о ячейках вся информация записывается в конец микролога. При переполнении микролога происходит копирование неустаревшей информации из одной ячейки в другую, очищение рабочей ячейки, после чего ячейки меняются ролями.

Структуры данных в оперативной памяти

При монтировании драйвера файловой системы flashfs происходит считывание информации о содержащихся в системе файлах и их расположении из описанных таблиц, расположенных во flash-памяти устройства. В отличие от JFFS при монтировании надо просматривать не все ячейки flash-памяти, а только несколько, что ускоряет процесс монтирования. Из каждого микролога выбираются и записываются в память только корректные значения. Информация из таблицы файлов используется для заполнения структур vfs. Можно сконфигурировать файловую систему таким образом, что таблица трансляций будет представлена в оперативной памяти массивом. Отдельно хранится список свободных блоков flash-памяти.

4.4.2 Особенности реализации ramfs

Файлы в ramfs представляют собой нефрагментированные линейно расположенные именованные области данных. Структура файловой системы создается в оперативной памяти при каждой загрузке системы. Информация о файлах хранится в двусвязном списке, содержащем элементы следующей структуры:

```
typedef struct _ramfs_file_description {
        unsigned long start_addr;
        unsigned int  size;
        unsigned int  mode;
        unsigned int  mtime;
        int           cur_pointer;
        int           lock;
} ramfs_file_description_t;
```

Кроме информации о начальном адресе файла и его размере в этой структуре (файловом дескрипторе) содержится дополнительная информация о правах доступа, блокировках и т.п.. Выделение памяти происходит только при наличии свободного нефрагментированного куска требуемой длины. Информация о свободных кусках памяти системы хранится как отдельный список.

4.5 Доступ к файлам

В оперативной памяти системы расположен массив открытых файлов. Он содержит набор дескрипторов файлов, которые в данный момент открыты. Каждый дескриптор содержит информацию о начальном адресе файла, его размере и драйвере обслуживающей его файловой системы. При открытии файла происходит его поиск в файловой системе, в частности определяются все эти параметры. Это означает, что к открытому файлу можно обращаться, используя прямую ссылку на файл, без его повторного поиска в vfs.

В тех случаях, когда данные из flash-памяти используются в режиме только для чтения, имеется возможность организовать кеш этих файлов в оперативной памяти. В частности, если размеры оперативной памяти позволяют, можно все необходимые файлы из flash-памяти скопировать в оперативную память при загрузке системы.

При попытках организовать детерминированный доступ к файлам возникает проблема блокировок при доступе к файлам из разных задач. В текущей реализации при попытке доступа к заблокированному файлу, задача не зависает в заблокированном состоянии, а за фиксированное время получает сообщение о невозможности доступа к файлу. Это не решает проблему блокировок в общем случае, но позволяет в некоторых задачах добиться детерминированного времени работы системы.

4.6 Драйвера устройств

На нижнем уровне файловой системы располагаются драйвера устройств flash-памяти. На данный момент в проекте Embox поддерживаются NOR микросхемы flash-памяти Intel серии P30.

Заключение

Тема разработки файловых систем для встраиваемых систем сейчас очень актуальна. Как внутри различных компаний, так и opensource сообществом постоянно ведутся и публикуются исследования по данной тематике. Ознакомившись с этими наработками, я попытался решить задачу создания файловой системы в рамках своей предметной области.

Описанная в работе файловая система реализована в рамках проекта Embox. Она предоставляет универсальный интерфейс доступа к файлам, поддерживает работу с flash и ram памятью. Простота принципов построения делает ее небольшой по объему (~10 файлов порядка 100 строк кода без драйверов устройств) и в то же время приемлемой для использования. В данный момент ведутся работы по ее улучшению, в первую очередь в области быстродействия и соответствия стандарту POSIX. Кроме того, ведется работа по добавлению новых драйверов файловых систем.

В соответствие с принципами ОСРВ Embox все части файловой системы организованы в виде отдельных независимых модулей, что позволяет сконфигурировать ее в минимально необходимой в каждом конкретном случае варианте.

Файловая система ОСРВ Embox нашла свое применение в проектах компании ЗАО «Ланит-Терком» по разработке встраиваемых систем, в том числе решающих задачи реального времени робототехники.

Список литературы

1. Файловая система OCPB Embox. Режим доступа: http://code.google.com/p/embox/wiki/FileSystemDescription
2. David Woodhouse, JFFS: The Journaling flash file system. Режим доступа: http://sourceware.org/jffs2/jffs2-html/
3. Theodore Tso, SSDs, Journaling and noatime/relatime, Режим доступа: http://www.linuxfoundation.org/news-media/blogs/browse/2009/03/ssd%E2%80%99s-journaling-and-noatimerelatime
4. М. Тим Джонс, Анатомия ext4. Режим доступа: http://www.ibm.com/developerworks/ru/library/l-anatomy-ext4/
5. М. Тим Джонс, Анатомия виртуального коммутатора файловых систем Linux. Режим доступа: http://www.ibm.com/developerworks/ru/library/l-virtual-filesystem-switch/index.html
6. М. Тим Джонс, Анатомия файловой системы Linux. Режим доступа: http://www.ibm.com/developerworks/ru/library/l-linux-filesystem
7. М. Тим Джонс, Анатомия файловых систем Linux для флэш-носителей. Режим доступа: http://www.ibm.com/developerworks/ru/library/l-flash-filesystems
8. Э. Таненбаум, Современные операционные системы. 2-е изд. – СПб.: Питер, 2007. – 1038 с.: ил.
9. Э. Таненбаум, А. Вудхалл, Операционные системы. Разработка и реализация. 3-е изд. – СПб.: Питер, 2007. – 704 с.: ил.

10. Coffee filesystem guide. Режим доступа: http://www.sics.se/contiki/wiki/index.php/Coffee_Filesystem_Guide
11. POSIX, Portable Operating System Interface for Unix, режим доступа: http://standards.ieee.org/develop/wg/POSIX.html
12. Режим доступа: http://www.testfreaks.com/blog/information/usb-flash-drive-comparison-part-2-fat32-vs-ntfs-vs-exfat
13. Режим доступа: http://sites.google.com/site/gouffs/home