САНКТ - ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет
Кафедра системного программирования

Колянов Дмитрий Андреевич

Использование внешнего устройства с энерго-независимой памятью для сохранения образа кэша в системе хранения данных

Магистерская диссертация

Допущена к защите.
Зав. Кафедрой:
д. ф.-м. н., профессор А. Н. Терехов

Научный руководитель:
д. ф.-м. н., профессор В.М. Нестеров

Рецензент:
О. В. Субботин

Санкт-Петербург
2013

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics&Mechanics Faculty
Chair of Software Engineering

Kolyanov Dmitry

Using external device with non-volatile memory to store cache image in storage array

Master's Thesis

Admitted for defence.
Head of the chair:
Dr. of Phys. and Math. Sci. A. N. Terekhov

Scientific supervisor:
Dr. of Phys. and Math. Sci. V. M. Nesterov

Reviewer:
O. V. Subbotin

Saint-Petersburg
2013

Содержание

1 Введение

Организация хранения информационных ресурсов является важной задачей для современных компаний, поэтому востребованы средства, гарантирующие сохранность информации. В связи с этим существует выбор большого числа систем хранения данных — комплексных программно-аппаратных средств, которые дают возможность создавать и настраивать централизованное и масштабируемое хранилище.

В зависимости от размера и характера деятельности компании требования к системам хранения могут сильно отличаться: для предприятий с большим числом клиентов, которым одновременно нужен доступ к важной информации, например, к состоянию счетов, нужны сверх-надежные и быстрые системы, поскольку потеря данных или невозможность доступа к ним может повлечь за собой большие временные и финансовые потери. Напротив, для организации документооборота или почтового сервера для компании средних масштабов не требуется значительная производительность и надежность. То есть компании, относящиеся к малому и среднему бизнесу часто не имеют возможности создания дорогостоящей IT-структуры и в связи с этим существует потребность в системах с низкой стоимостью.

В данной работе рассматривается одна из конфигураций существующей системы хранения из нижнего ценового диапазона. Для повышения производительности на данной системе используется кэширование: часть оперативной памяти системы резервируется под кэш, в который попадают запросы на чтение и запись. Задача данной работы — предотвращение исчезновения содержимого кэша в случае потери питания или выхода системы из строя.

2 Постановка задачи

Для предотвращения потери содержимого кэша в системе используется специальная кэш-карта, которая представляет из себя внешнее PCI – устройство, независимую встроенную систему, на которой хранится синхронизированная копия кэша, то есть каждая запись в кэш дублируется записью в память кэш-карты.

Задачами данной дипломной работы являются:

- Поддержка кэш-карты на системе хранения:
  - реализация библиотеки для взаимодействия с кэш-картой на уровне команд;
  - реализация драйвера кэш-карты;
  - реализация библиотеки для доступа к драйверу;
  - создание имитационной модели кэш-карты.
- Реализация механизма сохранения и восстановления данных на кэш-карте;
- Реализация диагностики кэш-карты в режиме реального времени;
- Создание системы сборки для прошивки кэш-карты.

3 Обзор системы хранения

Рассматриваемая система хранения существует в двух вариантах: Single SP и Dual SP. SP (storage processor) – это центральная часть системы, аналог материнской платы. Каждый SP имеет соединение с массивом хранения (back-end) и с сетью (front-end). В оперативной памяти SP располагается зеркалируемый кэш.

В конфигурации Dual у системы есть два SP, которые связаны друг с другом через PCI-E. В случае выхода одного из них из строя система продолжит функционировать за счет второго. В этой конфигурации кэш присутствует на обоих SP и синхронизируется через PCI-соединение, то есть оба SP в любой момент времени имеют одинаковое содержимое кэша.

В конфигурации Single, соответственно, есть только один SP, а вместо второго используется кэш-карта (cache memory card) для защиты от потери кэша в случае внезапной полной неработоспособности SP. Она не имеет соединения с дисковым массивом и сетью, единственная связь с кэш-картой осуществляется через PCI-E. То есть в данной конфигурации каждая запись в кэш дублируется записью на кэш-карту, а в Dual конфигурации запись дублируется на второй SP. Single конфигурация менее надежна, но при этом имеет значительно меньшую стоимость. В данной работе речь идет только о Single конфигурации, архитектура которой будет подробнее рассмотрена далее.

SP и кэш-карта являются заменяемыми модулями, поэтому система может быть восстановлена после сбоя путем замены одного из них. Содержимое кэша в этом случае может быть прочитано из кэш-карты, а уже сохраненные данные из массива хранения.

3.1 Конфигурация Single SP и кэш-карта

Кэш-карта представляет из себя встроенную систему на базе процессора PowerPC. Основные ее компоненты — это процессор, оперативная память и flash память. Flash память используется в качестве репозитория для кэша. При потере связи с SP данные из оперативной памяти сохраняются на flash память. Когда SP сможет продолжить функционирование, данные при необходимости могут быть восстановлены с кэш-карты.

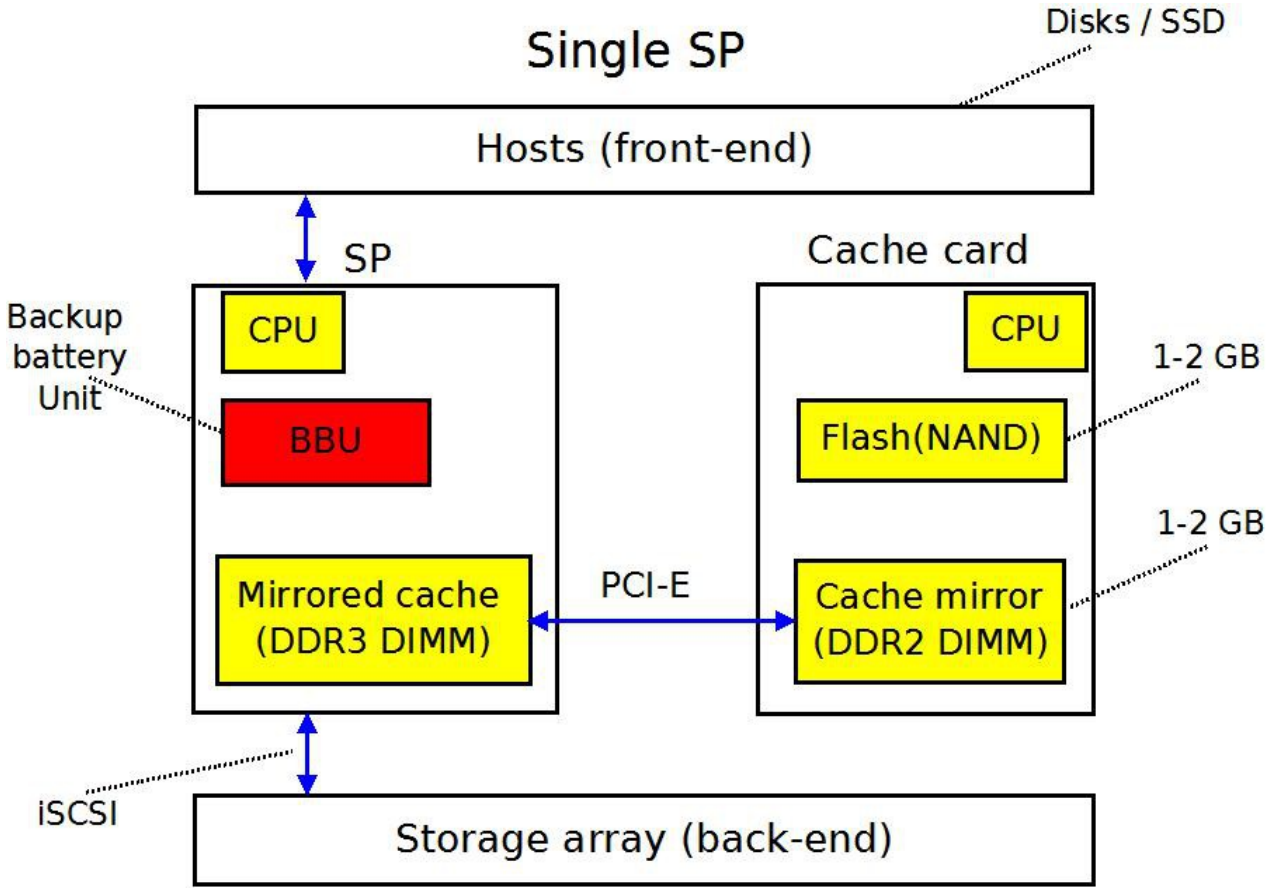

Рис. 1 Single SP конфигурация

При сбое работа кэш-карты обеспечивается за счет источника бесперебойного питания, установленного на SP. Для того, чтобы сохранить данные, кэш-карте требуется определенное время, поскольку NAND память работает достаточно медленно. После сохранения кэша кэш-карта сигнализирует об этом на аппаратном уровне.

4 Архитектура

Все компоненты, связанные с кэшом реализованы на системе хранения в двух конфигурациях: помимо обычной сборки, каждая компонента может быть собрана в режиме симуляции. Симуляция требуется для разработки и тестирования — режим симуляции позволяет запускать весь требуемый программный стек вплоть до кэша при помощи специального тестового окружения. При этом не требуется наличие самой целевой системы, то есть того аппаратного обеспечения которое на ней установлено.

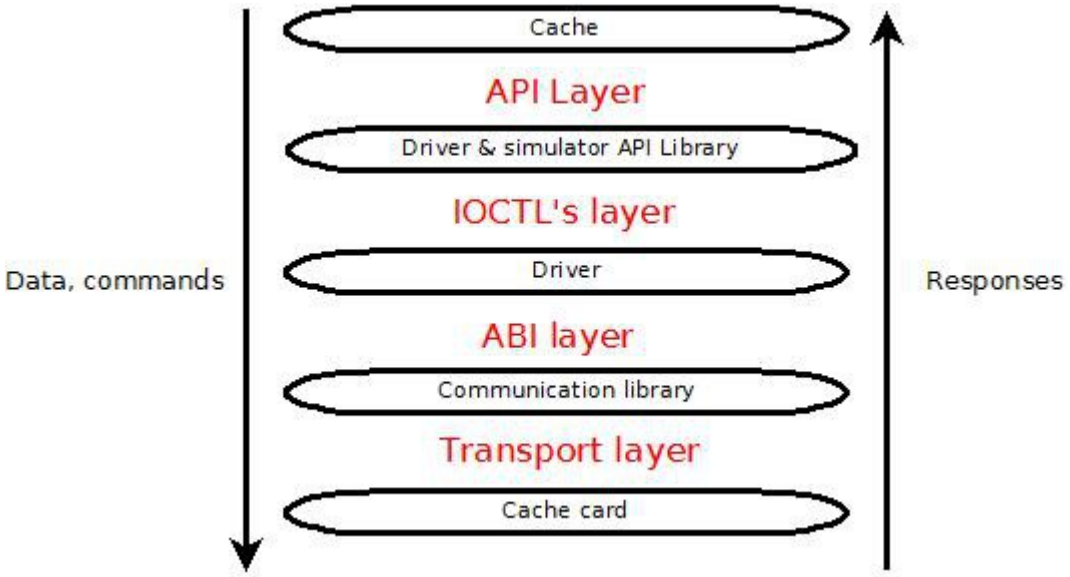

Рис. 2 Общая архитектура решения

Проведение тестов на системе не удобно по двум причинам: во-первых тестирование занимает значительно большее время, а во-вторых системы не всегда доступны для использования. В тестовом окружении написаны тесты для кэша и драйвера кэш-карты. Реализация симулятора кэш-карты будет рассмотрена отдельно.

Главной из рассматриваемых компонент является драйвер — он отвечает за установку соединения с кэш-картой, отправку команд, запись и чтение данных, обновление прошивки в случае необходимости, а так же наблюдение за

состоянием кэш-карты.

Интерфейс к драйверу — это библиотека с набором команд для кэш-карты. В нее также входят команды для управления симулятором. Кэш, инструменты командной строки и тесты используют данную библиотеку для взаимодействия с драйвером.

Коммуникация с кэш-картой производится при помощи двух циклических буферов, доступных для транзакций PCI-E. Для чтения и записи этих буферов используется коммуникационная библиотека: в ее задачи входит запись команд в нужном формате, их чтение, а так же отслеживание переполнения буферов, библиотека скрывает протокол общения с внешним устройством.

4.1 Библиотека API

В данном разделе будет описан интерфейс для взаимодействия с кэш-картой и симулятором. Библиотека предоставляет следующие функции для работы с кэш-картой:

- смена состояния автоматов кэш-карты:
  - autovault – состояние в котором внешнее устройство начинает сохранять данные из оперативной памяти на flash память;
  - ready – состояние, в котором запускается кэш-карта, в этом состоянии кэш-карта доступна для сброса, является переходным состоянием;
  - restore – состояние, в котором возможно восстановление с кэш-карты;
  - vault – состояние в котором начинается сохранение данных на кэш-карту.
- запись данных в оперативную память кэш-карты;
- сброс кэш-карты — перезагрузка операционной системы на кэш-карте;
- команда на восстановление данных в оперативную память;
- чтение из оперативной памяти кэш-карты;
- запрос данных о характеристиках кэш-карты (размер памяти и т.д.);
- запрос версии прошивки кэш-карты;
- запрос счетчиков ошибок в оперативной памяти;
- сброс счетчиков ошибок оперативной памяти;
- запрос счетчиков ошибок на flash памяти;
- обновление прошивки кэш-карты;
- запрос на очистку flash памяти;
- запрос на очистку оперативной памяти;

- установка размера кэша в оперативной памяти.

Дополнительно предоставляется несколько диагностических команд:

- тестирование скорости PCI транзакций;
- запрос внутренней диагностики кэш-карты;
- дамп буферов обмена;
- статистика операций чтения и записи;
- запрос объема свободной и используемой памяти на внешнем устройстве;
- запись файла в оперативную память кэш-карты;
- чтение файла из оперативной памяти кэш-карты;

Для симулятора работают все команды, доступные для кэш-карты. Библиотека предоставляет следующие дополнительные функции для работы с симулятором кэш-карты:

- симуляция потери соединения на кэш-карте;
- симуляция таймаута команды;
- установка конфигурации симулятора;
- симуляция ошибок flash памяти;
- симуляция ошибок оперативной памяти.

4.2 Драйвер кэш-карты

При старте драйвер кэш-карты устанавливает связь с драйвером PCI. Он регистрируется как клиент и выставляет соответствующие обработчики событий: обработчик потери соединения на уровне PCI и обработчик завершения DMA транзакции.

Запись на уровне PCI производится при помощи списков сборки-рассеяния (scatter-gather list), которые содержат указатели на участки физической памяти и их длину, драйвер предоставляет функции-обертки, которые формируют нужные списки и ожидают подтверждение окончания транзакции в течение заданного времени. Функции записи могут быть асинхронно использованы несколькими клиентами, для этого драйвер организует очередь с заранее выделенными структурами для запроса.

Для взаимодействия с кэш-картой драйвер выделяет два буфера – in и out и хранит указатели на них.

Логика наблюдения за состоянием кэш-карты разбита на две компоненты, каждая из которых выполнена в виде конечного автомата.

Первая компонента (transport) отвечает за установку и поддержку соединения с кэш-картой. Для идентификации кэш-карты используется пара из заранее определенной константы (magic) и версии протокола. Драйвер является инициатором соединения и в первую очередь запрашивает вышеописанную пару значений. Если пара совпадает с имеющимися данными в драйвере, автомат продолжает работу по установке соединения. В результате транспортный автомат начинает непрерывный обмен возрастающей последовательностью чисел (heartbeats) с картой. Обмен должен происходить с определенной частотой и последовательность чисел должна возрастать. Если последовательность не возрастает в течение заданного времени, то соединение считается прерванным.

Так как системы контролируются разными процессорами, фактическая частота обмена на каждой стороне может несколько отличаться. Для стабилизации разницы в отсчете времени в обмен включается время системы при отправке очередного числа последовательности (timestamp).

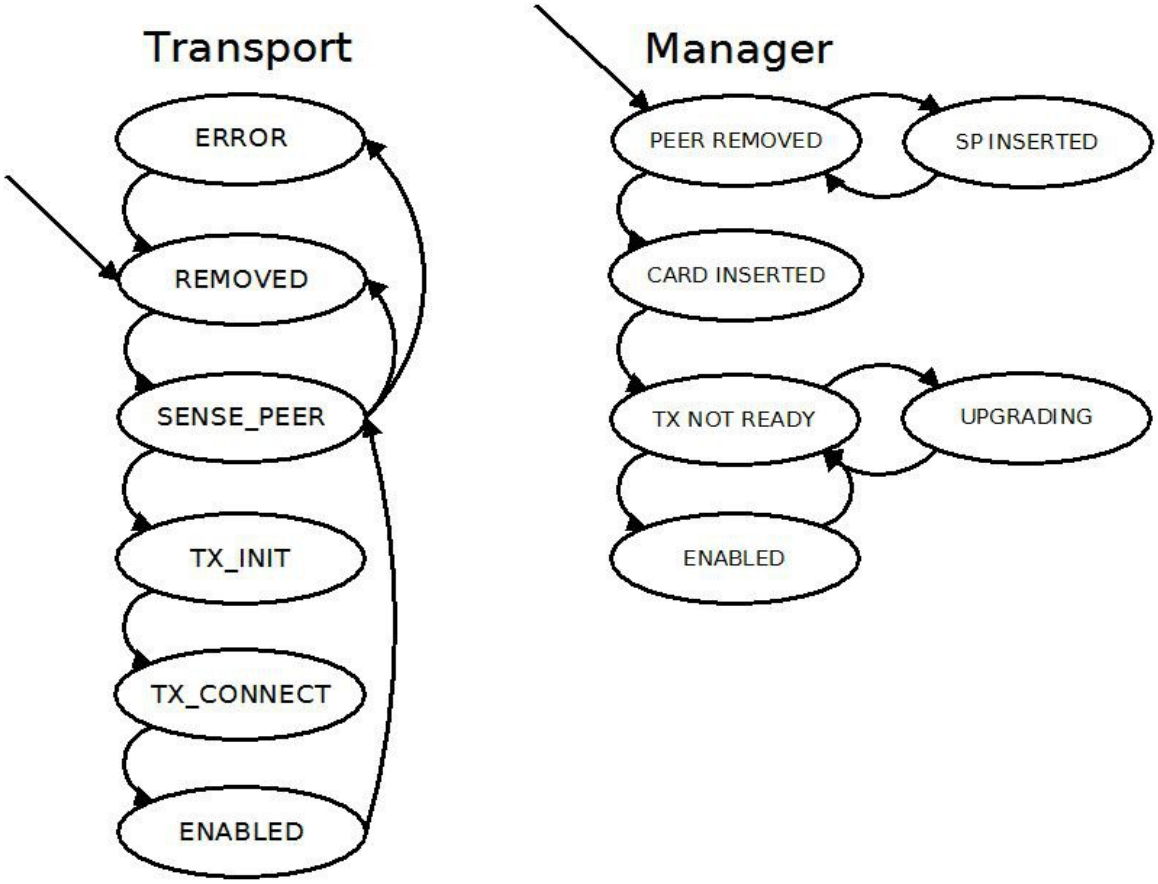

Рис. 3 Конечные автоматы драйвера

Рассмотрим подробнее состояния транспортного автомата:

- Removed – это состояние является стартовым, в нем осуществляется проверка наличия кэш-карты или второго SP в системе;
- Sense peer – состояние в котором определяется соседнее устройство — SP или кэш-карта;
- TX Init – состояние в котором инициализируются начальные данные для обмена;
- TX Connect – состояние, отвечающее за логику установки соединения, в

случае неудачи автомат пытается установить соединение снова;

- Enabled – состояние, в котором производится наблюдение за соединением, в случае его потери автомат пытается заново установить соединение;
- Error – состояние в котором вместо кэш-карты в системе находится SP.

Таким образом транспортный автомат либо поддерживает соединение, либо непрерывно пытается его восстановить и при наступлении определенных событий посылает уведомление ведущему автомату. Такими событиями могут являться установка/потеря соединения или исчезновение кэш-карты из системы.

Основная функция ведущего автомата (manager) — отображать актуальное состояние кэш-карты. В зависимости от состояния автомат позволяет или запрещает отправку команд и запись данных на кэш-карту.

Ведущему автомату предоставлена возможность сброса или перезагрузки кэш-карты, что по сути является практически единственным способом устранения неисправностей на ней. Поэтому при получении соответствующего события от транспортного автомата, ведущий меняет свое состояние и пытается перезагрузить кэш-карту. Он же контролирует количество попыток восстановления.

Перед входом в состояние enabled автомат проверяет актуальность прошивки кэш-карты. Для этого он находит последнюю доступную версию прошивки на системе, запрашивает версию прошивки с кэш-карты и при необходимости делает обновление.

Рассмотрим подробнее состояния ведущего автомата:

- Peer removed – в системе нет ни кэш-карты ни второго SP, запись запрещена;
- SP inserted – в системе есть второй SP, запись запрещена;
- Card inserted – в системе есть кэш-карта, но соединение с ней еще не

установлено, запись запрещена;

- Tx not ready – соединение потеряно, запись запрещена;
- Upgrading – система в состоянии обновления прошивки, запись запрещена.
- Enabled – есть соединение, на кэш-карте установлена последняя версия прошивки, запись разрешена, автомат начинает периодически опрашивать кэш-карту на предмет внутренних ошибок.

За отправку команд отвечает отдельная компонента, которая для каждой команды записывает данные в out буфер в нужном формате, используя при этом коммуникационную библиотеку, ждет ответ в течение определенного времени и обрабатывает полученные данные.

Появление ответа в in буфере отслеживается в транспортном автомате в цикле состояния enabled. При поступлении данных автомат разбирает заголовок сообщения и вызывает соответствующий обработчик, который сигнализирует о поступлении ответа процессу, пославшему команду.

4.3 Коммуникационная библиотека

Для отправки и получения команд выделено два циклических (ring) буфера со стороны системы и два таких же буфера выделены со стороны кэш-карты. Первый буфер называется «IN» и используется только для чтения, второй буфер называется «OUT» и используется только для записи.

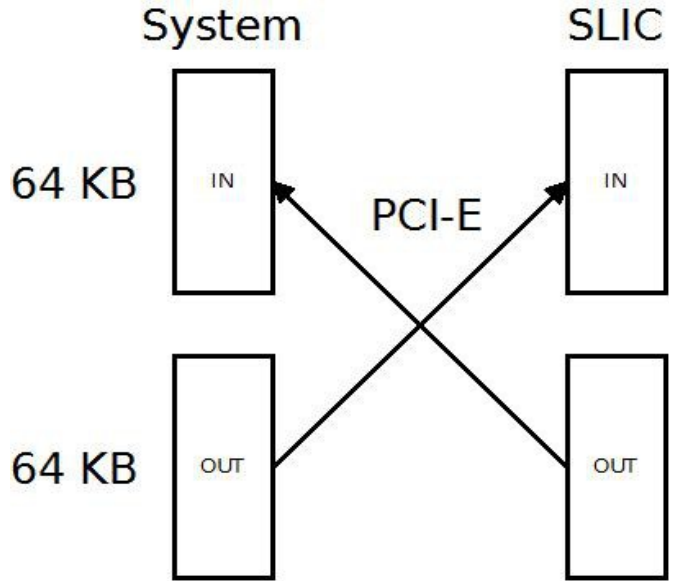

Рис. 4 Схема взаимодействия системы и кэш-карты

Каждый буфер разбит на три области:

- Link area — область, в которую записываются данные, необходимые для установки и поддержки соединения;
- Transport area — область в которую записываются команды;
- Control area — область в которую записываются данные, нужные для контроля позиций чтения и записи в транспортной области, а также отслеживания переполнения буферов.

Рис. 5 Структура IN/OUT буфера

При записи команды в транспортную область данные выравниваются по

размеру блока данных — 16 B. Control area содержит два поля: read_pos и write_pos, которые указывают на то откуда производится чтение и куда производится запись на данной стороне. То есть в IN буфере содержится указатель на последнее место записи в IN буфер, и последнее место чтения из OUT буфера, в OUT буфере содержится указатель на последнее место записи в OUT буфер и указатель на последнее место чтения из IN буфера. Таким образом для контроля позиций чтения и записи достаточно четырех указателей.

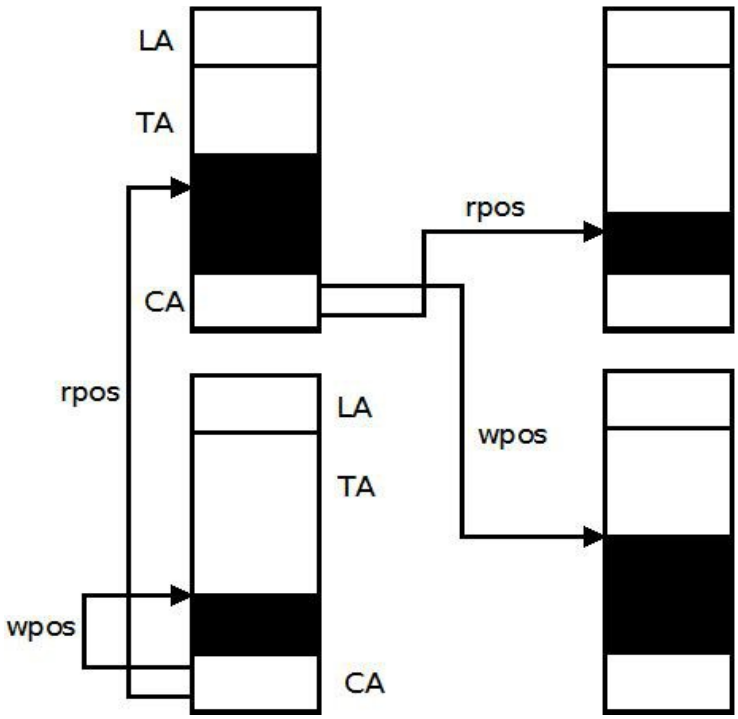

Рис. 6 Позиции чтения и записи

Порядок записи байт на системе и кэш-карте отличается, поэтому библиотека при передаче сообщения использует сетевой порядок, а при получении - порядок, определяемый архитектурой принимающей стороны.

Сообщения могут быть двух типов: в виде строки, состоящей из пар ключ-значение и в виде бинарных данных(структуры). Каждому сообщению предшествует заголовок, в котором содержится размер сообщения, его тип, а так же номер сервиса, который должен его обработать.

Библиотека предоставляет интерфейс для записи каждого сообщения в транспортную область, а так же функции для работы с control и link областями.

4.4 Симулятор кэш-карты

Для симуляции кэш-карты написан отдельный модуль, который частично повторяет ее поведение. Модуль представляет из себя отдельный процесс, который взаимодействует с симуляционной версией драйвера.

При старте модуль инициирует собственные буфера in и out, которые он выделяет в разделяемой памяти (shared memory). Драйвер в симуляционной версии делает то же самое. Каждый регион разделяемой памяти имеет свое имя, что позволяет процессам находить регионы друг друга. Операцией отправки данных служит не PCI-транзакция, а операция копирования.

В целом симулятор имеет подобную драйверу архитектуру. Основной его частью является обработчик команд драйвера, который получает запросы, обрабатывает их и отправляет на них ответ. Для поддержки соединения используется ведомый транспортный автомат, который ожидает инициации соединения, а после поддерживает его. В качестве памяти для записи кэша выступает файл.

Симулятор, как и драйвер, может принимать внешние команды. В библиотеке API реализован интерфейс для работы с ним. Симуляция активно используется в тестах кэша и драйвера, реализованный модуль позволяет имитировать все возможные команды драйвера, присутствие или отсутствие кэш-карты в системе, потерю соединения, различные ошибки на кэш-карте, задержку выполнения команд, различные виды конфигурации кэш-карты и т.д.

5 Архитектура кэш-карты

Прошивка кэш-карты состоит из четырех частей и хранится на загрузочном flash устройстве:

- Загрузчик (u-boot)
- Ядро (linux kernel) — основная часть прошивки.
- Device tree — структура, которую заполняет загрузчик и передает ядру — содержит информацию об устройствах, например их физических адресах, регистрах, размере, и т.д.
- Rootfs (squashfs) — корневая файловая система, содержит набор утилит, необходимых для разработки.

Все изменения из данной работы, касающиеся логики прошивки относятся только к ядру, поэтому остальные компоненты рассматриваться далее не будут.

Адресное пространство ядра ограничено 32 мегабайтами, остальная оперативная память использована для зеркалирования кэша. В ядре реализован драйвер, в основе которого находится несколько конечных автоматов, два из которых отвечают за соединение с SP, а третий является обработчиком команд основной системы.

У кэш-карты есть аппаратный интерфейс, позволяющий управлять ее перезагрузкой, com-порт для взаимодействия с кэш-картой при разработке, а так же интерфейс *JTAG* для низкоуровневой отладки.

5.1 Сохранение и восстановление данных

Сохранение данных производится на NAND flash память. Flash память разбита на физические блоки, которые в свою очередь разбиты на страницы. Минимальной единицей чтения и записи является страница, но Flash память позволяет стирать ее блоками. Каждый блок имеет достаточно большой ресурс, однако со временем возможно появление так называемых bad блоков — блоков не пригодных для записи.

Flash память распознается ядром linux как mtd устройство. На уровне mtd решаются задачи физической записи страниц на flash память. Mtd предоставляет интерфейс чтения/записи страниц, на этом уровне производится подсчет контрольных сумм, есть функции для проверки физического блока на валидность, а так же маркировки его как bad.

Для эффективной работы с flash устройствами необходим логический уровень. В качестве логического уровня в данном случае выступает модуль UBI – система управления томами. На этом уровне память разбивается на логические блоки, каждому из которых может соответствовать какой-либо физический блок, причем с произвольным номером. В каждый блок записываются данные, необходимые для управления томами, счетчики записей, номер логического блока и другая служебная информация, которая занимает из соображений производительности одну или две страницы.

На уровне UBI решается задача равномерного распределения нагрузки на каждый блок (wear-leveling), то есть число записей в каждый физический блок с течением времени будет примерно одинаковым, хотя в логические блоки запись может идти как угодно. При записи или стирании может появиться bad block, поэтому UBI берет на себя функцию по их обработке, при попадании в такой блок во-первых он будет помечен как bad, а во-вторых будет произведен поиск нового физического блока для записи. При чтении данных могут возникать sbe

(single bit error), поэтому UBI обрабатывает и такие ситуации. Блок с sbe отправляется на тестирование и если он его не проходит, то помечается как bad.

Основной драйвер использует функциональность UBI и может работать с несколькими возможными конфигурациями flash памяти:

- 1 device x 8Gib;
- 2 devices x 8 Gib;
- 2 devices x 4 Gib;

В последней конфигурации время записи существенно больше чем в предыдущих из-за аппаратных особенностей устройств. Поэтому для повышения производительности был реализован механизм страничного чередования записи (interleaving).

У каждого устройства flash памяти есть страничный буфер куда первоначально попадают данные, а затем, после специальной команды, эти данные программируются на само устройство. Запись данных в буфер и программирование занимают определенное время, поэтому была применена следующая схема:

1. Запись данных в буфер устройства 1;
2. Команда программирования устройства 1;
3. Запись данных в буфер устройства 2;
4. Команда программирования устройства 2;
5. Ожидание завершения программирования устройства 1;
6. Ожидание завершения программирования устройства 2;

Асинхронная схема была реализована на уровне драйвера кэш-карты, UBI и mtd, что в результате дало 70% прирост производительности при записи.

5.2 Диагностика ошибок оперативной памяти

Система хранения может непрерывно работать в течение нескольких лет, поэтому повышается вероятность возникновения ошибок в оперативной памяти, то есть неконтролируемой смены значений отдельных битов. Вся память обычно разбивается на участки, защищенные ecc-кодами. Если изменяется один бит на участок, то ошибка называется sbe(single bit error) – она может быть обнаружена и исправлена. Если меняется несколько бит на одном участке — то ошибка называется mbe(multiple bit error) – она может быть обнаружена, но исправлена быть не может.

Ошибки могут быть обнаружены только при чтении участка памяти: в случае их обнаружения процессор генерирует соответствующие прерывания. В ядре Linux существует кросс-платформенный драйвер (EDAC), реализующий обработчики прерываний, задача которых — исправить ошибку. Процессор кэш-карты поддерживает пару инструкций при помощи которых возможно исправление ошибки в памяти, таким образом, что данные в памяти останутся правильными в независимости от того сколько процессов и обработчиков прерываний одновременно обращается к данному участку.

Драйвер обеспечивает обнаружение ошибки только при непосредственном обращении к памяти, но для уменьшения вероятности возникновения mbe нужно периодически читать всю память кэш-карты. Поэтому в драйвере была дополнительно реализована очередь отложенных действий, с помощью которой ядро раз в день читает всю память. Информация об ошибках сохраняется в виде счетчиков, которые доступны основному драйверу кэш-карты.

Помимо этого был реализован интерфейс sysfs, при помощи которого можно генерировать mbe и sbe ошибки (error injection).

5.3 Диагностика ошибок flash памяти

Flash память подвержена износу, в ней могут появляться bad блоки. Такие блоки непригодны для записи, поэтому объем памяти с течением времени может уменьшаться.

UBI предоставляет возможность решать эту задачу следующим образом: часть flash памяти резервируется под обработку bad блоков. То есть при возникновении bad блока на его место будет взят блок из резервного пула, размер пула при этом уменьшится, а объем доступной пользователю памяти останется прежним.

В драйвере кэш-карты была реализована команда, которая запрашивает информацию о состоянии flash памяти у UBI. На основе полученных данных о размере резервного пула, драйвер на стороне SP может принять решение о необходимости замены кэш-карты.

В основном драйвере кэш-карты производится последовательная запись данных на flash память и последующее их чтение и сравнение. В случае несовпадения прочитанных и записанных данных или в случае превышения таймаута на запись кэш-карта увеличивает счетчик ошибок flash памяти. SP периодически опрашивает счетчик ошибок и при необходимости драйвер может попытаться восстановить кэш-карту путем удаления и создания на ней UBI разделов или при неудачной попытке восстановления отрапортовать о том что кэш-карта нуждается в замене.

6 Результаты

В ходе данной работы были получены следующие результаты:

- Реализованы программные компоненты для поддержки кэш-карты:
  - библиотека для взаимодействия с кэш-картой на уровне команд;
  - драйвер кэш-карты;
  - библиотека для доступа к драйверу;
  - симулятор кэш-карты;
- реализован механизм сохранения и восстановления данных на кэш-карте;
- получено 70% повышение производительности записи;
- реализована диагностика кэш-карты в режиме реального времени;
- реализована система сборки для прошивки кэш-карты.

В результате данной дипломной работы были решены все поставленные задачи, ряд предложенных оптимизаций позволил снизить стоимость и повысить эффективность системы. Разрабатываемые программные компоненты успешно прошли тестирование и Single конфигурация системы была выпущена в производство.

7 Литература

[1] Роберт Лав. Разработка Ядра Linux, 2-е издание / Лав, Роберт. - М.: ООО «И.Д. Вильямс», 2008. - 448 с.

[2] Linux Device Drivers, Third Edition By Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. O'Reilly Media, Inc. 2005. - 636 p.

[3] MPC8544E PowerQUICC™ III Integrated Host Processor Family Reference Manual, (http://www.freescale.com/files/32bit/doc/ref_manual/MPC8544ERM.pdf)

[4] PowerPC™ e500 Core Family Reference Manual, (http://www.freescale.com/files/32bit/doc/ref_manual/E500CORERM.pdf)

[5] 4Gb D-die NAND Flash Datasheet,

(http://guimli.free.fr/download/K9F4G.pdf)