

# geom(4)

## НАЗВАНИЕ

GEOM -- модульный механизм преобразования запросов дискового ввода-вывода.

## ОПИСАНИЕ

Механизм GEOM (GEOM framework) предоставляет инфраструктуру, в которой "классы" могут выполнять преобразования запросов дискового ввода-вывода по пути от ядра до драйверов устройств и обратно.

Преобразования в контексте GEOM могут быть в диапазоне от простого геометрического перемещения, выполняемого в типичных модулях разбиения диска на разделы (disk partitioning modules) по алгоритмам RAID, и множественного разрешения имени устройства, до полной криптографической защиты сохраняемых данных.

По сравнению с традиционным "управлением томами", GEOM отличается от большинства, а, в некоторых случаях, и всех прежних реализаций следующими характеристиками:

- Механизм GEOM - расширяемый. Написать новый класс преобразований - просто до тривиальности, и он станет полноправным компонентом механизма. Если кому-то по некоторой причине захочется монтировать дисковые пакеты IBM MVS, создать класс, распознающий и конфигурирующий их *виртуальное содержание* (VTOS information), будет элементарно.
- Механизм GEOM не привязан к определенной топологии. Большинство реализаций управления томами очень строго определяют возможное взаимодействие классов, и очень часто предоставляется единственная фиксированная иерархия, например: поддиск - сплетение - том.

Расширяемость означает, что новые преобразования обрабатываются точно так же, как и существующие.

Фиксированные иерархии - это плохо, потому что они не позволяют эффективно выразить требуемую конфигурацию. В представленной выше фиксированной иерархии нельзя зеркаливать два физических диска, а затем разбить зеркало на поддиски. Вместо этого, придётся создавать поддиски на каждом физическом томе, а затем объединять их в зеркало, что дает намного более сложную физическую конфигурацию. Механизму GEOM, с другой стороны, безразличен порядок выполнения преобразований, - не допускаются только циклы в графе.

## ТЕРМИНОЛОГИЯ и ТОПОЛОГИЯ

Механизм Geom - существенно объектно-ориентированный, поэтому терминология формально и семантически во многом позаимствована из словаря объектно-ориентированного подхода:

"Класс", представленный структурой данных **g\_class**, реализует один определенный вид преобразования. Типичные примеры - классы для раздела диска MBR, метки диска BSD и массива RAID5.

Экземпляр класса называется "geom" и представляется структурой данных **g\_geom**. В типичной системе FreeBSD на платформе i386 будет один **geom** класса MBR для каждого диска.

"Поставщик", представленный структурой данных **g\_provider**, - это внешний интерфейс, через который **geom** предоставляет свои услуги. Поставщик - это "аналог диска, представленный специальным файлом в каталоге **/dev**", другими словами, - логический диск. У всех поставщиков есть три основных свойства: имя, размер сектора и общий размер.

"Потребитель" - это "черный ход", через который **geom** подключается к поставщику другого **geom**, и через который посылаются запросы ввода-вывода.

Между этими сущностями есть следующие топологические взаимоотношения:

- У класса есть ноль или более экземпляров **geom**.
- Каждый **geom** является экземпляром ровно одного класса.
- У каждого **geom** есть ноль или более потребителей.
- У каждого **geom** есть ноль или более поставщиков.
- Потребитель может быть подключен к одному поставщику или не подключен вообще.
- К поставщику может быть подключено сколько угодно потребителей.

У всех объектов **geom** есть числовой ранг, используемый для выявления и предотвращения циклов в направленном ациклическом графе. Этот ранг присваивается следующим образом:

1. Объект **geom** без подключенных потребителей получает ранг 1
2. Объект **geom** с подключенными потребителями получает ранг на единицу больше, чем максимальный ранг объектов **geom**s, к поставщикам которых подключены его потребители.

## СПЕЦИАЛЬНЫЕ ТОПОЛОГИЧЕСКИЕ МАНЕВРЫ

Помимо непосредственного присоединения потребителя к поставщику и отключения, нарушающего связь, есть еще ряд специальных топологических маневров, упрощающих конфигурирование и увеличивающих общую гибкость.

*Проба* (TASTING) - это процесс, происходящий при создании нового класса или нового поставщика и дающий классу шанс автоматически сконфигурировать экземпляры для поставщиков, которые он распознает как свои. Типичный пример - класс разбиения диска на разделы MBR, который будет искать таблицу MBR в первом секторе и, если находит и успешно проверяет ее допустимость, будет создавать экземпляр **geom** для мультиплексирования в соответствии с содержимым MBR.

Новый класс будет поочередно представляться всем существующим поставщикам, а новый поставщик - всем существующим классам.

Что конкретно должен делать класс, определяя, надо ли ему принять представленного поставщика, не задается механизмом GEOM, но следующий набор вариантов точно имеет смысл:

- Проверить конкретные структуры данных на диске.
- Проверить свойства поставщика, например, размер сектора и размер носителя.
- Проверить значение ранга объекта **geom** поставщика.
- Проверить имя метода объекта **geom** поставщика.

*Отрыв* (ORPHANIZATION) - это процесс, с помощью которого поставщик удаляется, хотя потенциально еще используется.

Когда объект **geom** отрывает поставщика, все последующие запросы ввода-вывода будут отвергаться ("bounce") поставщиком с кодом ошибки, который устанавливает объект **geom**. Любые потребители, подключенные к поставщику, получат уведомление об отрыве, когда до них доберется цикл обработки событий, и в этом случае они могут выполнить соответствующее действие.

Объект **geom**, возникший в результате обычной пробы, должен самоуничтожиться, если только он не может продолжать работу при отсутствии оторванного поставщика. Объекты **geom** типа секций диска должны поэтому самоуничтожиться, а объекты **geom** типа RAID5 или зеркала смогут продолжить работу, если только им хватает для этого оставшихся компонентов.

Отрыв поставщика не обязательно приводит к немедленному изменению топологии: присоединенные к нему потребители остаются присоединенными, открытые файлы - открытыми, а еще не выполненные запросы ввода-вывода - не выполненными.

Типичный сценарий при этом такой:

- Драйвер устройства обнаруживает, что диск перестал работать и отрывает соответствующего поставщика.
- Объекты **geom**, использующие этот диск, получают событие отрыва и, в свою очередь, отрывают своих поставщиков. Поставщики, к которым никто не присоединен, при этом обычно сразу самоликвидируются. Это процесс продолжается квази-рекурсивным образом, пока все соответствующие фрагменты дерева не узнают о сбое.
- Процесс распространения информации о сбое прекращается, когда достигает объекта **geom\_dev** наверху стека.
- Объект **geom\_dev** вызовет **destroy\_dev(9)**, чтобы остановить поступление запросов. Он будет ждать, пока все еще не выполненные запросы ввода-вывода (если они есть), не закончатся. Затем он явно все закроет (т.е. обнулит счетчики обращений), распространив это изменение по всему соответствующему поддереву. После этого он отключит и уничтожит свой объект **geom**.
- Объект **geom**, поставщик которого подключен, уничтожит поставщик, отключит и уничтожит его потребителей и, наконец, уничтожит его объект **geom**.
- Этот процесс продолжается дальше, пока не завершится очистка.

Хотя такой подход кажется неоправданно вычурным (*byzantine*), он действительно обеспечивает максимальную гибкость и надежность обработки "исчезающих" устройств.

Важно учитывать один абсолютно принципиальный нюанс: если драйвер устройства не завершит все запросы ввода-вывода, дерево не будет очищено (в оригинале - *unravel*).

*Порча* (SPOILING) - специальный случай отрыва, используемый для защиты от устаревания метаданных. Суть отражения будет проще понять на примере.

Представьте себе диск, "da0", на базе которого **geom** класса MBR предоставляет секции "da0s1" и "da0s2", а на базе секции "da0s1" **geom** класса BSD предоставляет разделы от "da0s1a" до "da0s1e". Пусть объекты **geom** классов MBR и BSD были сконфигурированы автоматически на основе структур данных на носителях. Теперь представьте себе ситуацию, когда диск "da0" открыт на запись и эти структуры данных изменены или переписаны: теперь объекты **geom** будут работать с устаревшими метаданными, если только некая система уведомления не проинформирует их об этом.

Чтобы избежать этой ситуации, когда происходит открытие устройства "da0" на запись, все подсоединенные потребители уведомляются об этом, и объекты **geom** вроде MBR и BSD, в результате, самоуничтожаются. Когда устройство "da0" снова будет закрыто, оно будет предлагаться для пробы и, если структуры данных для MBR и BSD на диске еще есть, соответствующие объекты **geom** заново себя создадут.

Теперь окончательное описание:

Если был открыт любой файл, полный путь к которому проходит через модуль MBR или BSD, этот модуль должен был быть открыт с установкой исключительного бита, что не позволяет в этом случае открывать

устройство "da0" на запись, и наоборот, запрошенный исключительный бит не позволит открыть файл через объект **geom** класса MBR, пока устройство "da0" открыто на запись.

Из этого также следует, что изменение размера открытых объектов **geom** может быть выполнено только при их содействии.

Наконец, порча происходит только когда счетчик количества записей с нулевого становится не нулевым, а повторная проба - только когда счетчик количества записей с не нулевого становится нулевым.

*Вставка/удаление (INSERT/DELETE)* - это очень специальная операция, позволяющая создавать новый объект **geom** между потребителем и поставщиком, подсоединенными друг к другу, и снова удалять его.

Чтобы понять необходимость этого, представте себе поставщика, смонтированного как файловая система. Между потребителем объекта **geom** класса DEVFS и его поставщиком мы вставляем модуль зеркалирования, который конфигурируется с одной зеркальной копией и, следовательно, прозрачен для соответствующих запросов ввода-вывода. Теперь можно сгенерировать зеркальную копию объекта **geom**, обеспечивающего зеркалирование, запросить синхронизацию и, наконец, удалить первую зеркальную копию. Теперь мы, по сути, перенесли смонтированную файловую систему с одного диска на другой, пока она использовалась. В этот момент объект **geom**, обеспечивающий зеркалирование, можно удалить - он выполнил свою функцию.

*Конфигурирование (CONFIGURE)* - это процесс выдачи инструкций администратором конкретному классу для создания экземпляра. В данном случае, есть несколько способов выразить такое намерение: можно указать конкретного поставщика, с уровнем override, что вызовет, например, подсоединение модуля метки диска BSD к поставщику, который не показался подходящим в ходе пробы.

Наконец, *ввод-вывод (IO)* - это причина, по которой все это вообще делается: эта операция связана с посылкой запросов ввода-вывода по графу.

*Запросы ввода-вывода (I/O REQUESTS)*, представленные структурой **bio**, генерируются потребителем, ставятся на выполнение подсоединенными к нему поставщиком, и, когда будут выполнены, возвращаются потребителю. Важно понимать, что структура **bio**, проходящая через поставщик некоторого объекта **geom** не "выходит с другой стороны". Даже простые преобразования, вроде выполняемых классами MBR и BSD, будут клонировать структуру **bio**, изменять клон и ставить его на выполнение собственным потребителем. Учтите, что клонирование структуры **bio** не требует клонирования области фактических данных, указанной в запросе ввода-вывода.

Всего механизм GEOM поддерживает четыре разных запроса ввода-вывода: **read** ( чтение), **write** (запись), **delete** (удаление) и **get attribute** (получение атрибута).

Назначение запросов чтения и записи понятно из названия.

Удаление обозначает, что определенный диапазон данных больше не используется, и может быть стерт или освобожден, в зависимости от того, что поддерживает базовая технология хранения. Технологии вроде флэш-памяти могут решить стереть соответствующие блоки, прежде чем они снова будут выделены, а устройства с шифрованием могут заполнить диапазон случайными битами, чтобы уменьшить объем данных, по которым можно выполнить атаку.

Важно понимать, что обозначение удаления - это не запрос, и нет никакой гарантии, что данные фактически будут стерты или сделаны недоступными, если только это не гарантируется конкретными объектами **geom** в графе. Если необходима семантика "безопасного удаления", необходимо добавить объект **geom**, преобразующий обозначение удаления в последовательность соответствующих запросов записи.

Получение атрибута поддерживает проверку и обработку экстренных (out-of-band) атрибутов конкретного поставщика или пути. Атрибуты именуются текстовыми строками и будут описаны в соответствующем разделе далее.

(продолжение следует, когда у автора отдохнут мозги и пальцы)

## ХРОНОЛОГИЯ

Это программное обеспечение было разработано для проекта FreeBSD Полом-Хёнингом Кампом (Poul-Henning Kamp) и лабораторией NAI Labs, исследовательским подразделением по защите (Security Research Division) корпорации Network Associates, Inc. по контракту DARPA/SPAWAR N66001-01-C-8035 ("CBOSS"), в рамках исследовательской программы DARPA CHATS.

Первым предшественником механизма GEOM была отвратительная поделка в Minix 1.2, которая никогда не распространялась. Ранняя попытка реализовать менее универсальную схему в OC FreeBSD так и не закончилась успехом.

## АВТОРЫ

Пол-Хёнинг Камп (Poul-Henning Kamp) <[phk@FreeBSD.org](mailto:phk@FreeBSD.org)>

FreeBSD 4.9, 27 марта 2002 года

Copyleft (no c) - Fuck copyright! 2004 В. Кравчук, OpenXS Initiative, перевод на русский язык