Логические ошибки в подсистемах ввода-вывода современных операционных систем

А.Б. Бетелин[1], А.А. Прилипко[2], Г.А. Прилипко[3], С.Г. Романюк[4], Д.В. Самборский[5]

[1]ab@niisi.msk.ru, [2]aaprilipko@niisi.msk.ru, [3]prilipko@niisi.msk.ru, [4]sgrom@niisi.ras.ru, [5]samborsky_d@fastmail.com

[1,2,3,4,5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия

Аннотация. Современные операционные системы используют уровни абстракции для описания интерфейса ввода-вывода данных. Построенные по этим принципам операционные системы обычно соответствуют стандарту POSIX, описывающему синтаксис и семантику системных вызовов. Тем не менее, применение стандартов не гарантирует согласованности программных компонент и корректности использованных алгоритмов. В данной работе описано несколько логических ошибок, типичных для сложных систем, разные части которых разрабатывались независимо. Анализ описанных ошибок дает основания утверждать, что детальное описание программных моделей систем ввода-вывода для их формальной верификации и тестирования создаваемого кода могут значительно увеличить надежность операционных систем и прикладных программ.

Ключевые слова: файловая система, RAID-массив, логическая ошибка, формальная верификация, тестирование

1. Введение

Стандарт интерфейса POSIX операционных Unix-подобных систем описывает системный интерфейс операционной системы, в том числе операции с файлами и каталогами [1]. С момента своего создания более 30 лет назад стандарт POSIX в некоторой степени эволюционировал, но в целом изменения были небольшими. В стандарте POSIX так и не были описаны промежуточные уровни системы ввода-вывода, необходимые для построения различных систем хранения данных. Внутреннее устройство ядра операционной системы не было предметом стандартизации, поскольку задачей стандарта POSIX было создание общего интерфейса для ОС Unix разных производителей. Кроме того, на момент написания первых версий стандарта POSIX операционные системы семейства Unix были устроены значительно проще, чем современные ОС.

Например, типичная конфигурация файлового сервера ОС Linux включает в себя уровни файлового кэша (page cache), файловой системы (интерфейс virtual file system, VFS), системы управления логическими томами (logical volume management, LVM), программного RAID-массива и подсистем драйверов SCSI или NVMe. Соответственно, каждая операция записи данных проходит через стек этих уровней и может быть завершена как подтвержденной записью в устройства накопителей, так и записью в файловый кэш или в очередь отложенных операций драйверов блочного ввода-вывода. Прикладная программа имеет только ограниченные средства контроля за порядком выполнения этих операций, основные параметры подлежат настройке на системном уровне. Поэтому для настройки оптимальной производительности подсистемы ввода-вывода или диагностики возникающих в ней системных ошибок могут потребоваться глубокие знания об архитектуре ОС и специальные диагностические инструменты [2]. Описания алгоритмов работы разных уровней подсистемы ввода-вывода и драйверов устройств часто не перекрываются между собой и сводятся к техническому описанию деталей реализации, что усложняет понимание того, как работает ОС в целом в случае возникновения нештатных ситуаций.

Если бы проектирование ОС выполнялось с использованием формальных спецификаций, то каждый уровень программного интерфейса мог бы формулировать необходимые условия для входных параметров и, со своей стороны, гарантировать выполнение определенных обязательств. Когда эти условия и обязательства детально описаны, появляются возможности верификации — как с помощью формального доказательства недостижимости некорректных состояний, если алгоритмы тестируемых компонентов достаточно просты, так и с помощью функциональных тестов, имитирующих разнообразные штатные и аварийные ситуации.

Ниже приведены примеры логических ошибок и ситуаций неопределенного поведения в программных комплексах, которые были обнаружены на стадии эксплуатации, хотя могли быть обнаружены ранее, на стадиях проектирования и разработки.

2. Потеря данных в дисковых массивах RAID5/6

Чтобы обеспечить отказоустойчивость, массив RAID5 объединяет блоки данных в группы и дополняет каждую группу дополнительным блоком четности, в котором хранится результат побитовой операции XOR, суммирования по модулю 2 битов данных этих блоков. Блоки одной группы расположены на разных дисках, это дает теоретическую возможность полного восстановления данных при отказе одного из этих дисков. Аварийная ситуация ошибки записи, называемая также "write hole", возникает в массиве RAID5 в случае одновременного выхода из строя одного из дисковых устройств и аварийной перезагрузки системы во время записи данных. Предположим, что в случае минимального массива RAID5, состоящего из трех дисков, в блоках одной группы располагаются данные A1, B1 и A1+B1, соответственно, где A1+B1 обозначает данные блока четности с побитовой суммой данных блоков A1 и B1. Если в момент аварийного отключения были обновлены данные блока B, а блок четности обновиться не успел, то состояние данных блоков станет A1, B2 и A1+B1. При одновременном выходе из строя первого диска окажется, что данные A1 полностью потеряны. Действительно, попытка их восстановления будет выполняться по неверной формуле A1+B1+B2 вместо верной формулы A1+B2+B2=A1, которая бы применялась, если блок четности успел бы обновиться. Ситуация усугубляется, когда блок A1 содержит служебные данные файловой системы или другие важные данные, не связанные с адресатом записи. Потеря этих данных может быть обнаружена позднее, когда они понадобятся прикладным программам или операционной системе, поэтому ошибки такого вида называют скрытыми повреждениями данных или "silent data corruption". Стратегия журнализирования записей файловой системы в этом случае помочь не может.

Вероятность одновременного системного сбоя и выхода из строя одного из дисков на практике оказывается выше ожидаемой, поскольку эти события могут иметь общие причинно-следственные связи. Ошибки этого вида могут произойти и при большем количестве дисков, а также в случае использования RAID6.

Аналогичный сбой в массиве RAID1, т.е. в режиме простого дублирования данных, приведет только к потере последней записи. При этом выполняемая в момент сбоя операция записи не подтверждена, поэтому любая прикладная программа, ответственно подходящая к хранению данных (т.е. использующая синхронизацию записи вызовом fsync или выполняющая запись в синхронном режиме O_DSYNC), должна будет вызвать повторную запись. Целостность файловой системы при таком сбое обеспечит журнал, в котором будет храниться информация для восстановления последнего согласованного состояния структур данных.

Для предотвращения ошибок вида "write hole" в 2015 году ядро Linux было модифицировано: в исходном коде MD-драйвера, реализующего функцию дисковых массивов RAID5 и RAID6, была добавлена поддержка журнала записи [3] на отдельном устройстве. Таким образом, данная проблема была решена только в версии Linux 4.4 – через 15 лет после появления функции поддержки дисковых массивов RAID5 и RAID6. Позже, в версии Linux 4.12, был также добавлен механизм "partial parity log" (PPL), сохраняющий старые версии блоков данных и блоков четности в служебных структурах разметки дисков RAID-массива. Включение механизма PPL снижает производительность, в худшем случае на 30-40% [4], но зато предотвращает аварии вида "write hole" без необходимости создания журнала записи.

Пример этой ошибки демонстрирует, что если обновление данных становится хотя бы немного более сложным, чем простое копирование, как в RAID1, то попытки избежать журнализирования неизбежно приводят к потерям данных. На эту опасность теоретики систем управления базами данных указывали еще более 40 лет назад, см. "UPDATE IN PLACE: A poison apple?" [5].

3. Останов работы RAID1-массива в случае сбоя работы одного устройства

Учитывая вышеописанные опасности использования дисковых массивов RAID5/6, можно предположить, что уровни RAID1 и RAID10 обеспечат надежную и бесперебойную работу. Действительно, алгоритм работы на этих уровнях гораздо проще, поэтому они обеспечивают и сохранность данных, и повышенную скорость чтения. Тем не менее, программные RAID-массивы ядра Linux могут не оправдывать ожидания бесперебойной работы во всех случаях неисправности дисковых устройств. Алгоритм ра-

боты MD-драйвера подразумевает, что устройство либо выполнит операцию чтения или записи, либо сообщит об ошибке, но не остановит свою работу на неопределенное время. Т.е., если диск не ответил на запрос, то весь RAID-массив будет находиться в состоянии ожидания, что для пользователей хранилища данных будет выглядеть как отказ, который может привести к ошибкам в прикладных программах после истечения таймаутов, и, в конечном счете, к потере данных. Данная ситуация не была предусмотрена при разработке кода MD-драйвера, хотя она, очевидно, демонстрирует нестыковку уровней системного кода.

Было предложено два способа решения этой проблемы. Во-первых, можно обеспечить включение аппаратных таймаутов в устройствах накопителей, вызывающих завершение операции с кодом ошибки, чтобы программный уровень RAID-массива мог пометить это устройство как отказавшее и продолжить работу в неполном составе. Позже администратор может диагностировать это устройство, и либо заменить, либо повторно ввести в эксплуатацию. Для устройств с интерфейсом SATA/SAS, предназначенных для использования в серверах, такая настройка должна присутствовать (так называемые TLER, ERC, или CCTL таймауты). Во всех популярных дистрибутивах ОС Linux настройка этих таймаутов не выполняется автоматически при включении дисков в RAID-массивы, хотя соответствующая поправка к пакету smartmontools была предложена 6 лет назад [8]. Во-вторых, несколько позже в MD-драйвере для уровней RAID1 и RAID10 была предложена поддержка атрибута таймаута "failfast" [6, 7].

Эти меры могут исключить полный останов программного RAID-массива, однако просто необычно долгие задержки в работе устройств, не превышающие заданные таймауты, все же могут существенно снизить среднюю скорость работы ввода-вывода. В случае использования магнитных дисковых накопителей рекомендуется периодическая сверка копий данных RAID-массива, которая вызывает тестирование поверхности дисков и перенос сбойных секторов данных, которые иначе могли бы вызвать снижение скорости работы устройства. Для SSD-накопителей, показывающих нестабильное время выполнения операций, обычно нет других решений кроме обновления внутренней прошивки или замены более надежными и, как правило, более дорогими моделями накопителей.

Если дисковый массив используется для хранения данных виртуальных ОС, то среда виртуализации также могла бы использовать таймаут и, например, запланировать приостановку виртуальной системы в случае недоступности устройства. Тем не менее, многие популярные среды виртуализации не предоставляют такой возможности. Например, у QEMU/KVM есть атрибут error_policy для драйверов блочного хранилища, но он предусматривает только реакцию на ошибки ввода-вывода и ситуацию исчерпания свободного места в хранилище виртуальных дисков.

4. Неверная интерпретация семантики функций синхронизации сервером СУБД PostgreSQL

В марте 2018 года разработчики СУБД PostgreSQL обнаружили, что сбой записи данных на диск в ОС Linux может остаться незамеченным и привести к потере данных сервером баз данных [9]. Причиной этих ошибок было то, что семантика системных вызовов fsync и fdatasync не соответствовала ожиданиям разработчиков PostgreSQL. Эти системные вызовы нужны для принудительной синхронизации записи данных указанного файлового дескриптора, т.е. для сброса страниц файлового кэша и буферов ввода-вывода на постоянный носитель информации. Они используются, если файл открыт в обычном режиме (без опций O_DSYNC и O_DIRECT), и программе необходимо получить гарантию, что данные записаны на носитель и не будут потеряны при сбое или аварийном отключении сервера.

В случае неудачной попытки выполнить запись данных на носитель функция fsync возвращает код ошибки EIO. Но в документации ОС Linux не сообщается, что происходит с незаписанными блоками данных. Оказалось, что ядро ОС Linux в этом случае очищает данные, как будто они были записаны, и поэтому повторный вызов fsync для этого файлового дескриптора уже не сообщает об ошибке. Программа PostgreSQL в случае неудачи повторяла вызов fsync и, убедившись в его успешном завершении, считала данные записанными, а первую ошибку считала временным сбоем устройства. В ответ на запрос о причинах такого поведения разработчики ядра Linux пояснили, что данная функция сообщает только об ошибочных ситуациях, возникших после последнего, а не последнего успешного вызова fsync. Из этого следует, что повторный вызов fsync не имеет смысла в ОС Linux даже для тех накопителей данных, которые могут быть временно недоступны, а потом восстановиться после сбоя.

Такой алгоритм работы ядра ОС Linux был выбран потому, что чаще всего подобные ошибки вызывались неожиданным отключением USB флэш-накопителей, когда трудно ожи-

дать, что устройство будет возвращено и продолжит работу, а накопление незаписанных данных привело бы к переполнению оперативной памяти и сбою всей системы. Однако известно, что некоторые POSIX-совместимые системы, например, ОС FreeBSD, все же не делают такую очистку данных. Поэтому в ОС FreeBSD повторные вызовы fsync могут инициировать успешную запись данных, если устройство накопителя данных вернулось к нормальной работе, или продолжают возвращать код ошибки EIO, если устройство все еще не готово к приему данных.

Если переданные для записи данные были потеряны, то прикладной программе в среде ОС Linux остается либо повторить все операции записи с момента последнего успешного вызова fsync (т.е. вызовы функций write, writev, aio_write, и др.), либо аварийно завершить работу, тем самым передав проблему на более высокий уровень. Разработчики PostgreSQL выбрали второй вариант, поскольку при рестарте сервера СУБД будет прочитан журнал запланированных записей (write ahead log, WAL) и восстановлено последнее корректное состояние базы данных [10]. Если же ошибка EIO возникала во время записи в сам журнал WAL, то в этой ситуации и более ранние версии сервера PostgreSQL сразу сигнализировали об ошибке и завершали свою работу. Кроме того, операции записи в журнал WAL в среде ОС Linux по умолчанию выполняются в синхронном режиме, т.е. используется опция O_DSYNC вызова open, и поэтому вызовы fsync по отношению к ним не нужны.

Данный пример демонстрирует, что даже в предельно простой ситуации, когда в среде одной ОС работает одна прикладная программа и выполняет буферизованную запись на одно устройство, то все равно не удается обойтись без семантических неопределенностей на стыках уровней абстракций.

5. Алгоритм использования внутреннего кэша записи SSD-накопителей

Все модели SSD-накопителей серверного применения имеют область кэш-памяти, которая позволяет откладывать запись данных во flash-память, тем самым достигая высокой производительности даже для операций записи не выровненных и сильно фрагментированных данных. Алгоритм использования этой кэш-памяти также тесно связан с уровнем отображения логических блоков данных в страницы flash-памяти, поэтому такое кэширование необходимо не только для увеличения производительности, но и для экономии ограниченного ресурса перезаписи страниц памяти устройства накопителя.

При этом производители большинства моделей SSD-накопителей гарантируют сохранность всех отложенных для записи данных при внезапной потери энергопитания. Эта функция называется Power Loss Protection (PLP) и обеспечивается встроенной конденсаторной батареей, питающей накопитель в течении времени, необходимого для записи данных кэша в энергонезависимую flash-память.

Такая функция фактически стирает различие между кэш-памятью и flash-памятью. Поэтому некоторые модели SSD-накопителей игнорируют команды синхронизации кэша записи, которые драйвер накопителя посылает в ответ на системные вызовы fsync, инициированные прикладной программой. Другие же модели, наоборот, в ответ на эти команды скрупулезно выполняют запись всех отложенных для записи данных. По-сути, оба варианта поведения имеют смысл, поскольку есть свобода интерпретации смысла системного вызова fsync. Второй вариант предпочтителен, когда нужно гарантировать, что данные не будут потеряны, даже если накопитель затем подвергнется временному сбою по какой-либо причине — это может быть сбой микропрограммы, импульс в цепи электропитания или другое внешнее воздействие ("космические лучи"). Но обычно пользователь накопителя подразумевает, что ему достаточно гарантии сохранности данных при неожиданной потере электропитания. В этом случае для сохранения высокой производительности приложений с частыми вызовами fsync приходится либо использовать только накопители первого типа, либо пробовать программно отключать во внутренних настройках накопителя режим кэширования записи. Тогда накопитель, скорее всего, продолжит выполнять запись данных сначала в кэш-память, поскольку так работает его внутреннее ПО, но драйвер накопителя в ядре операционной системы перестанет посылать накопителю команды сброса данных, потому как будет уверен, что кэширование записи выключено. В результате вызовы fsync перестанут замедлять работу накопителя, т.к. будут игнорироваться.

Если накопитель не имеет функции PLP, то выключение кэширования сильно снизит скорость записи, поскольку тогда не только вызовы fsync, но и обычные операции записи будут ожидать подтверждения записи в микросхемы энергонезависимой памяти. Такие устройства не рекомендуется использовать без дополнительного HBA-адаптера или RAID-контроллера, т.к. при внезапной потере питания в них могут повредиться любые данные, даже не затронутые последними операциями записи. Используемый

HBA-адаптер или RAID-контроллер должен иметь собственную конденсаторную батарею с емкостью, достаточной для выполнения отложенных записей.

Таблица 1. Классификация SSD-накопителей по их заявленным характеристикам и измеренной производительности операций записи

Тип	Функция PLP	Скорость записи при вкл. кэше	Скорость записи при выкл. кэше	Комментарий
1	+	высокая	высокая	Устройство считает, что функция PLP гарантирует надежную запись
2	+	высокая	низкая	Устройство считает, что выключение кэширования требует записи данных во flash-память даже при наличии функции PLP
3	–	высокая	высокая	Устройство игнорирует выключение кэширования, несмотря на отсутствие функции PLP
4	–	высокая	низкая	Устройство не имеет функции PLP и корректно реагирует на выключение кэширования

Оказывается, что для достижения как надежности хранения, так и высокой скорости записи данных помогает следующее эмпирическое правило: нужно выключить кэширование записи в настройках устройства накопителя, измерить производительность операций записи, и если скорость снизилась, то либо использовать другие модели накопителей, либо подключить существующие через адаптер или контроллер, имеющий защиту от потери питания. В последнем случае настройку всех устройств следует выполнять согласно инструкции адаптера или контроллера. Таблица 1 классифицирует накопительные устройства согласно наблюдаемой производительности при включенном и выключенном кэшировании записи. Вышеупомянутое эмпирическое правило не работает только по отношению к устройствам третьего типа, которые скрывают отсутствие функции PLP, и уже поэтому должны быть исключены из рассмотрения. Таким образом, устройства первого типа допустимо использовать непосредственно, а устройства второго и четвертого типов рекомендуется подключать к HBA-адаптеру или RAID-контроллеру.

Для устройств второго типа такое подключение может считаться избыточным, но оно позволяет делегировать функцию кэширования контроллеру, что обычно повышает надежность и производительность. Дополнительно рекомендуется проверить скорость записи больших блоков данных при выключенном кэшировании записи в SSD-накопителе. Если скорость записи в этом режиме экстремально низкая, то это означает, что функция кэш-памяти необходима для нормальной работы устройства. Такие устройства не следует использовать даже при установке в RAID-контроллер, поскольку контроллер может настаивать на выключении кэша записи устройства накопителя.

В тех случаях, когда пользователям необходима надежность записи данных не только при потере электропитания, но и в иных ситуациях (отказ внутреннего ПО, единичный сбой микросхемы контроллера устройства), следует использовать устройства второго типа и выключить кэш записи. Это снизит скорость записи данных, но даст гарантии целостности записанных данных непосредственно после подтверждения операции. В этом случае гарантируется целостность завершенных транзакций записи, но не гарантируется выполнение всех транзакций. Если требуется дальнейшее увеличение степени надежности, то этого можно достичь дублированием устройств записи и обеспечением восстановления верной копии данных программными или аппаратными средствами (например, RAID-массивом).

6. Заключение

Из четырех вышеописанных ситуаций только

первая — потеря данных в массивах RAID5 — ограничена одним программным модулем. В остальных примерах причиной ошибочного или неожиданного поведения ОС и прикладных программ служит или недостаточно точно описанная программная модель подсистемы ввода-вывода, или ее неверная интерпретация, как в случае ошибки СУБД PostgreSQL. Это демонстрирует то, что в любых сложных программных системах могут скрываться ошибки, даже если эти системы имеют большую пользовательскую базу и успешно эксплуатируются в течение десятилетий. Решить эти проблемы могла бы формальная верификация корректности всего программного кода операционной системы, но для больших систем это практически невозможно. Так, пока единственным примером успешной верификации кода ОС является микроядро seL4 [11], имеющее всего 10 тысяч строк исходного кода.

Кроме ошибок системного ПО еще одним источником аварийных ситуаций служит непредсказуемость поведения оборудования. В этих условиях программистам прикладного и системного ПО следует подвергать сомнению спецификации как программных, так и аппаратных интерфейсов, и, при возможности, проверять их соблюдение специальными тестами. Архитекторам программных систем можно рекомендовать строить программные и аппаратные системы из проверенных комбинаций компонентов, т.е. не полагаться на теоретически совместимые, но не проверенные на практике конфигурации.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (Проведение фундаментальных научных исследований (47 ГП) по теме «1021060909180-7-1.2.1 Развитие методов математического моделирования распределенных систем и соответствующих методов вычисления. (FNEF-2022-0007)»).

Logical errors in the input/output subsystems of modern operating systems

A.B. Betelin, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. Modern operating systems use abstraction layers to describe data input/output interfaces. Operating systems based on these principles conform to the POSIX standard, which describes syntax and semantics of system calls. However, use of standards alone does not guarantee the consistency of used software components and correctness of designed algorithms. This article describes several logical errors which are typical for complex systems, different parts of which were developed independently of each other. An analysis of the described errors allows us to state that definition of the detailed models for software components and their subsequent formal verification and application for code testing can significantly improve robustness of the created operating systems and applications.

Keywords: file system, RAID array, logical error, formal verification, testing

Литература

1. Стандарт POSIX.1-2017. The Open Group Base Specifications Issue 7, 2018 edition IEEE Std 1003.1-2017.

2. Gregg, Brendan. Linux Systems Performance. Portland, OR: USENIX Association, 2019.

3. Сайт "A journal for MD/RAID5", https://lwn.net/Articles/665299

4. Сайт "Partial Parity Log for MD RAID 5", https://lwn.net/Articles/715280

5. J. Gray, et al. The transaction concept: Virtues and limitations, VLDB. – 1981. – T. 81. – C. 144-154.

6. Сайт "Add 'failfast' support for raid1/raid10", https://lwn.net/Articles/706865

7. Сайт "Timeout Mismatch", https://raid.wiki.kernel.org/index.php/Timeout_Mismatch

8. Сайт "Many (long) HDD default timeouts cause data loss or corruption (silent controller resets)", https://www.smartmontools.org/ticket/658

9. Сайт "PostgreSQL's handling of fsync() errors is unsafe and risks data loss at least on XFS", https://lwn.net/Articles/752093

10. Сайт «Руководство PostgreSQL. Часть III. Администрирование сервера. Глава 29. Надёжность и журнал предзаписи», https://postgrespro.ru/docs/postgresql/13/wal-reliability

11. Klein, Gerwin, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser. Comprehensive formal verification of an OS microkernel. ACM Transactions on Computer Systems (TOCS) 32, no. 1 (2014): 1-70.