УДК 004.451

АНАЛИЗ ЭФФЕКТИВНОСТИ ПРИМЕНЕНИЯ МЕТОДОВ ХЕШИРОВАНИЯ В ФАЙЛОВЫХ СИСТЕМАХ, РАБОТАЮЩИХ В ПОЛЬЗОВАТЕЛЬСКОМ РЕЖИМЕ

Е.Ю. Иванов, М.С. Косяков

Рассмотрены особенности протоколов взаимодействия виртуальной файловой системы и файловой системы, их влияние на производительность микроядерных операционных систем. На примере файловой системы ext2, реализованной для операционной системы MINIX 3, установлено, что время идентификации файлового объекта в микроядерных операционных системах может увеличиваться до 26 раз по сравнению с монолитными системами. Показано, что при использовании рекомендованных в работе методов хеширования можно добиться производительности рассматриваемой части стека ввода–вывода, практически не уступающей производительности монолитных операционных систем.

Ключевые слова: файловая система, операционная система, микроядерная ОС, методы хеширования, протоколы взаимодействия ВФС и ФС, MINIX 3, PUFFS.

Введение

Несмотря на достижения современных исследований в области надежности операционных систем (ОС) и верификации программного обеспечения, отказы ОС, вызванные аппаратными и программными ошибками, по-прежнему приводят к снижению доступности прикладных сервисов, в том числе в системах стратегического назначения [1]. По некоторым оценкам, в ОС на каждую тысячу строк кода приходится до шести ошибок [2, 3]; размер же современных ОС превышает 6 000 000 строк кода. Согласно [4], до 85% ошибок находится в коде драйверов устройств и файловых систем (ФС), причем в случае монолитных ядер ОС любая из них потенциально может привести к краху всей системы. Считается, что переход к модульным микроядерным ОС, в которых основной функционал реализован в виде системных процессов, работающих в непривилегированном пользовательском режиме, позволяет решить данную проблему [5].

Отказоустойчивость микроядерных ОС основана на принципе минимальных привилегий и изоляции ошибок в системных компонентах (модулях), в частности, в драйверах и ФС: при отказе одного из компонентов ядро ОС продолжает функционирование и в состоянии перезапустить сбойный системный процесс. Время восстановления одного компонента значительно меньше, чем время восстановления всей

системы, благодаря чему повышается коэффициент готовности. Кроме того, такая структурная организация ОС позволяет увеличить время наработки на отказ, так как с точки зрения прикладного сервиса отказ драйвера или ФС с последующим перезапуском соответствующего компонента в большинстве случаев может остаться вообще незамеченным.

Функционирование драйверов и ФС в виде отдельных процессов пользовательского режима приводит к потере производительности микроядерных ОС по сравнению с монолитными ОС [6]. Одной из причин снижения производительности является отсутствие возможности совместного использования служебных структур данных различными компонентами ОС. В микроядерных ОС системные процессы вынуждены явным образом обмениваться данными, используя средства ядра ОС, и осуществлять преобразование данных из одного формата в другой при их передаче согласно специальным протоколам взаимодействия. В частности, подобные накладные расходы появляются в микроядерных ОС из-за выделения виртуальной файловой системы (ВФС) и файловой системы в независимые процессы. При этом эффективность механизма их взаимодействия становится основным фактором, определяющим производительность этой части стека ввода–вывода в сравнении с монолитными ОС.

С целью увеличения производительности микроядерных ОС в данной работе проанализирована эффективность применения различных методов хеширования в ФС, работающих в пользовательском режиме, и рассмотрена целесообразность предоставления ВФС дополнительной информации о структурах данных, поддерживаемых ФС, как, например, в протоколе PUFFS взаимодействия ВФС и ФС для ОС NetBSD [7]. Для этого в микроядерной ОС MINIX 3 авторами реализована ФС ext2, что позволило, кроме прочего, провести сравнение производительности рассматриваемой части стека ввода-вывода в микроядерных ОС с аналогичными решениями для монолитных систем, а именно с ФС ext2, работающей в ОС Linux [8]. Показано, что при использовании рекомендованных в работе методов хеширования можно добиться производительности рассматриваемой части стека ввода–вывода микроядерных ОС, практически не уступающей производительности монолитных ОС.

Структурная организация стека ввода–вывода

На рис. 1 показана структурная организация стека ввода-вывода в монолитных и микроядерных ОС. При обращении к файлу пользовательский процесс выполняет системный вызов, передавая ОС дескриптор файла *fd* (*file descriptor*). Для монолитных ОС система переводится в режим ядра, а управление передается ВФС, которая по *fd* находит объект *vnode* (*virtual node*), содержащий высокоуровневое описание файла. Среди прочего, *vnode* предоставляет указатель на структуру данных ФС *istruct* (*inode struct*), описывающую сам файловый объект. ВФС использует этот указатель на *istruct* при вызове соответствующего обработчика запроса ФС. Таким образом, ВФС идентифицирует запрашиваемый файловый объект посредством прямого доступа к *istruct*. Данное описание применимо к Linux [8] и к большинству других UNIX-подобных ОС.

В микроядерных ОС ВФС и ФС представляют собой отдельные независимые процессы, работающие в пользовательском режиме. Как и в монолитной системе, ВФС использует *fd* для поиска соответствующего *vnode*. Затем, согласно протоколу взаимодействия ВФС и ФС, ВФС осуществляет запрос к ФС, используя некоторый идентификатор файла *id*. Использование в качестве *id* непосредственно адреса *istruct* является нежелательным, так как в этом случае нарушается принцип изоляции, и ФС не сможет перемещать *istruct* в памяти между вызовами со стороны ВФС. Тем не менее, некоторые протоколы взаимодействия ВФС и ФС, например, протокол PUFFS, реализованный для ОС NetBSD, в целях повышения производительности позволяют ВФС хранить и передавать адреса структур данных *istruct*, поддерживаемых ФС [7].

Другие протоколы, такие как протокол взаимодействия ВФС и ФС в ОС MINIX 3, в качестве *id* используют числовой идентификатор *inum*, получаемый от ФС при создании или открытии файла [9]. Использование идентификатора *inum*, отличного от адреса *istruct*, приводит к необходимости хранения в ФС таблицы соответствий *inum* адресам *istruct* и поиска по ней при идентификации файлового объекта. Так как эта операция используется при каждом обращении к ФС, ее неэффективная реализация может сказаться на производительности файловой системы. Стоит отметить, что данная проблема отличается от поиска *vnode* по значению файлового дескриптора *fd* в ВФС. В последнем случае количество хранимых *fd*, среди которых осуществляется поиск, соответствует числу файлов, открытых одним вызывающим процессом, т.е. относительно невелико. По этой причине операция поиска *vnode* по *fd* практически не оказывает влияния на время обслуживания запроса в ВФС. Поиск же адреса *istruct* по *inum* осуществляется среди всех идентификаторов файлов, открытых всеми процессами. Кроме того, ФС обычно кэширует сами структуры *istruct* на случай повторного обращения к недавно закрытым файлам, что приводит к дополнительному увеличению пространства поиска и, как следствие, времени выполнения запроса.

В связи с этим в работе проводится анализ эффективности применения различных методов хеширования при реализации таблицы соответствий *inum* адресам структур данных *istruct*. Измеряемой характеристикой является среднее время выполнения эталонных задач, представляющих собой интегральные

нагрузки, широко используемые для оценки производительности стека ввода–вывода ОС [10]. В качестве характеристики производительности используется величина, обратная времени выполнения, что соответствует числу задач, исполненных в единицу времени.

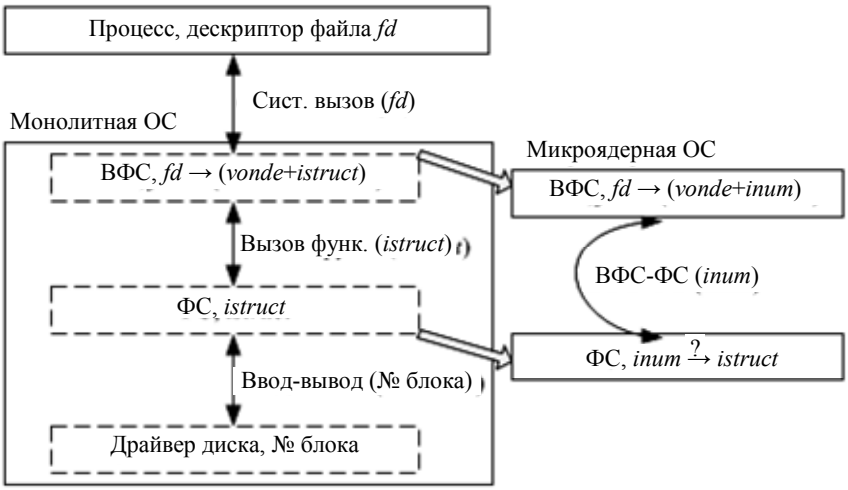

Рис. 1. Структурная организация стека ввода-вывода в монолитных и микроядерных ОС

Параметры проведения экспериментов

Эксперименты проводились на реализованной в работе файловой системе ext2 для микроядерной ОС MINIX 3, функционирующей на персональном компьютере с процессором Intel Pentium 4 2.80 ГГц, оперативной памятью DDR 512 МБ и диском Seagate 40 ГБ 7200 rpm. Рассматривались следующие типы эталонных пользовательских нагрузок:

- «Сборка». Сборка ОС MINIX 3 из каталога */usr*/*src*, включая ядро, драйверы, ФС и библиотеки. Полное время выполнения этой задачи *T* составляло около 40 мин.
- «Копирование». Копирование дерева каталогов с исходными файлами */usr*/*src* на целевую файловую систему ext2. Общий объем данных составлял порядка 170 МБ, число файлов 24 694 (1 620 каталогов), средняя глубина каталогов равнялась четырем. Время *T* равнялось примерно 20 с.
- «Статистика». Вызов системной команды *stat* для всех файлов из каталога */usr*/*src* в случайном порядке. В отличие от нагрузки «Копирование», в результате которой происходят последовательные обращения к диску, в нагрузке «Статистика» обращения к блокам диска осуществляются в произвольном порядке. Время *T* равнялось примерно 50 с.

Медленные дисковые операции значительно более сильно влияют на производительность стека ввода–вывода, чем вычислительные операции внутри ФС, что затрудняет измерение времени τ поиска адреса *istruct* по числовому идентификатору файла *inum*. Кроме того, при учете времени выполнения дисковых операций резко увеличивается длительность проведения экспериментов. В связи с этим авторами, в том числе, анализировалась работа указанной части стека ввода–вывода изолированно без учета дисковых операций. Функционирование такой части осуществлялось в специальном программном окружении, имитирующем остальные компоненты ФС. Для этого случая нагрузка на исследуемую часть стека ввода–вывода создавалась путем воспроизведения записанных ранее операций ФС, сформированных при выполнении перечисленных выше пользовательских задач.

С целью повышения точности измерений функция Hyper-Threading процессора была отключена. Процессу, исполняющему код исследуемой части стека ввода–вывода, был присвоен наивысший абсолютный приоритет, что исключало прерывание его работы для предоставления процессорного времени другим процессам. Подкачка страниц виртуальной памяти была запрещена. В качестве *inum* и адреса *istruct* в протоколе взаимодействия ВФС и ФС в ОС MINIX 3 используются целые 32-разрядные числа, поэтому их обработка на 32-битном процессоре не требует дополнительных накладных расходов.

Анализ эффективности методов хеширования

Результаты экспериментов показали, что при линейном поиске адреса *istruct* по *inum* и при ограничении максимального числа *n* хранимых структур *istruct* значениями до 512 время идентификации файлового объекта τ увеличивается до 26 раз по сравнению со временем, наблюдаемым при прямой адресации, используемой в монолитных системах. Для увеличения производительности рассматриваемой части стека ввода–вывода рассмотрена эффективность применения различных методов хеширования при реализации таблицы соответствий *inum* адресам структур данных *istruct*.

Производительность хеш-таблиц напрямую зависит от выбора хеш-функции, метода разрешения коллизий и размера хеш-таблицы. В работе использованы эталонные реализации хеш-таблиц на основе открытой адресации с квадратичной последовательностью проб (Google dense) и двойным хешированием (GNU libiberty). Для оценки эффективности хеш-таблиц с разрешением коллизий при помощи цепочек использована собственная реализация, позволяющая, в отличие от других, полностью контролировать все параметры вычисления. Например, в популярной реализации *hash_map*, входящей в GNU GCC, размер хеш-таблицы динамически изменяется в зависимости от коэффициента заполнения α. Кроме того, результаты экспериментов показали, что собственная реализация работает быстрее, чем *hash_map* при аналогичных параметрах.

Для отображения множества возможных значений *inum* в *m*-элементное множество хеш-значений, где *m* – размер хеш-таблицы, используются следующие хеш-функции *h*:

- модульное хеширование «MOD»: $h(inum) = inum \bmod m$. Применяется в GNU libiberty и *hash_map*;
- модульное хеширование «AND» для случаев, когда *m* является степенью двойки: $h(inum) = inum \; and \; (m - 1)$. Данная функция проста в вычислительном плане и дает распределение, схожее с распределениями модульных хеш-функций общего вида [11]. Используется в Google dense и в авторской реализации хеш-таблицы.

Собственная реализация хеш-таблицы, кроме хеш-функции «AND», поддерживает хеш-функцию «MOD», мультипликативное и универсальное хеширование, популярную хеш-функцию «MurmurHash» [12]. Наши результаты показали, что, несмотря на бо́льшую вычислительную сложность, перечисленные функции не дают заметного преимущества в распределении хеш-значений по сравнению с хеш-функцией «AND».

В таблице приведено среднее время τ идентификации файлового объекта при реализации различных типов файловых нагрузок для случая $n = 512$. Размер хеш-таблиц для Google dense и реализации авторов $m = 2048$, для libiberty $m = 2039$. Разница между наблюдаемыми минимальными и максимальными значениями τ составляла не более 7% от представленных средних значений.

Тип нагрузки	**Таблицы соответствий *inum* адресам *istruct***				
	Google dense	**libiberty**	**Авторская реализация**	**Прямая адресация**	**Линейный поиск**
«Сборка»	354	410	214	165	4315
«Копирование»	21	24	11	9	237
«Статистика»	10	13	6	5	104

Таблица. Среднее время идентификации файлового объекта τ, мс

Согласно полученным результатам, для случая работы с файлами разрешение коллизий на основе цепочек работает быстрее, чем открытая адресация. Преимущество метода «цепочек» сохраняется также при изменении размеров хеш-таблиц. Это объясняется высокой интенсивностью открытия и закрытия файлов при выполнении нагрузок, что требует частого удаления элементов из таблицы соответствий. При использовании открытой адресации удаленные элементы продолжают храниться в хеш-таблице, тем самым увеличивая время поиска. Более низкая производительность открытой адресации с двойным хешированием по сравнению с квадратичной последовательностью проб указывает на то, что время, затрачиваемое на вычисление дополнительных хеш-значений, превышает потери времени на дополнительных пробах [13]. Кроме того, видно, что при использовании авторской реализации хеш-таблицы с разрешением коллизий при помощи цепочек время идентификации файлового объекта лишь немногим превосходит соответствующее время, наблюдаемое при прямой адресации, используемой в монолитных системах. В этой связи в дальнейшем в работе рассматривается именно авторская реализация хеш-таблицы.

На рис. 2 показана зависимость времени τ от размера *m* хеш-таблицы для нагрузки «Сборка» при ограничении на число хранимых *istruct* $n = 512$. Графики, представленные пунктиром, получены при кэшировании последнего найденного сопоставления *inum* адресу *istruct*, что позволяет реже выполнять поиск в таблице соответствий. При значениях $m \geq 4096$, соответствующих коэффициенту заполнения $\alpha = n / m \leq 0{,}25$, значение τ становится практически идентичным результатам, полученным при использовании прямой адресации. Эксперименты показали, что данная зависимость справедлива и для других значений *n* при $\alpha \leq 0{,}25$. Кроме того, при таких значениях коэффициента заполнения α выигрыш от кэширования результатов последней операции поиска становится совсем незначительным. Перечисленные утверждения остаются верными и для других файловых нагрузок, так как идентификация файлового объекта необходима при выполнении любой файловой операции.

Результаты измерения полного времени *T* выполнения пользовательских задач с учетом дисковых операций показали, что при небольших $n \leq 1024$ значения $T^{\text{ЛП}}$, получаемые для случая линейного поиска адреса *istruct* по *inum* со средним временем $O(n)$, практически совпадают со значениями $T^{\text{ХТ}}$ для случая применения хеш-таблиц со средним временем поиска $O(1)$. Это объясняется тем, что медленные диско-

вые операции оказывают значительно б*о*льшее влияние на время T, чем вычислительные операции внутри ФС. Однако при увеличении числа n хранимых в памяти *istruct* отношение времен $T^{\text{ЛП}}/T^{\text{ХТ}}$ выполнения пользовательских задач для этих случаев также возрастает и достигает двух раз уже при $n \approx 4096$.

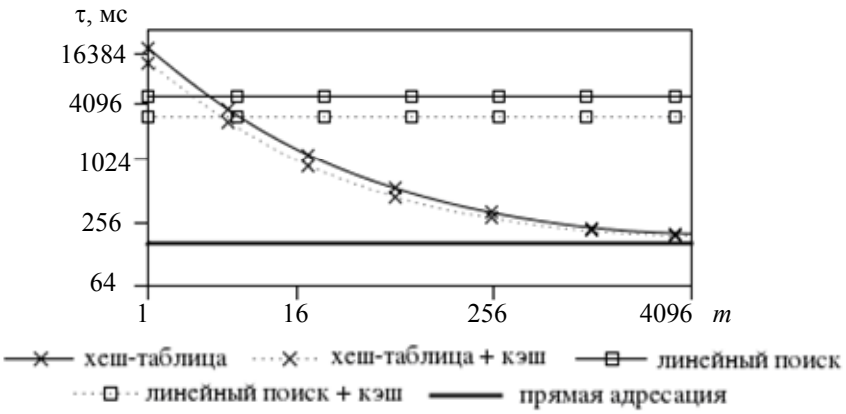

Рис. 2. Зависимость среднего времени идентификации файлового объекта т от размера хеш-таблицы m на примере нагрузки «Сборка»

Заключение

В работе рассмотрены особенности протоколов взаимодействия ВФС и ФС и их влияние на производительность микроядерных ОС, обладающих повышенной надежностью в сравнении с монолитными системами. Для детального изучения этой части стека ввода–вывода была разработана реализация файловой системы ext2, функционирующая в микроядерной ОС MINIX 3. С ее помощью было показано, что протоколы взаимодействия ВФС и ФС, использующие числовой идентификатор *inum* для поиска файловой структуры *istruct*, описывающей запрашиваемый файловый объект, привносят значительные накладные расходы. А именно, время идентификации файлового объекта увеличивается до 26 раз по сравнению с монолитными системами, что особенно заметно при рассмотрении работы ФС без учета дисковых операций, например, при доступе к блокам файла, находящимся в кэше ФС.

В качестве возможного пути решения этой проблемы в работе рассматривается подход, реализованный в протоколе PUFFS взаимодействия ВФС и ФС для ОС NetBSD, позволяющий ВФС хранить адреса структур данных, поддерживаемых ФС, в частности адреса *istruct*. К сожалению, данный подход нарушает принцип изоляции компонентов микроядерных ОС и ведет к существенной зависимости функционирования ВФС от работы ФС, тем самым понижая надежность ОС.

В этой связи в работе предлагается использовать альтернативный путь увеличения производительности рассматриваемой части стека ввода–вывода, позволяющий сохранить изоляцию ВФС от ФС и основанный на применении хеш-таблиц в ФС для поиска адресов *istruct* по *inum*. Результаты измерений времени идентификации файлового объекта при использовании различных методов хеширования показали допустимость применения простой в вычислительном плане модульной хеш-функции при размере хеш-таблицы $m = 2^x$. Было установлено, что для рассматриваемого случая работы с файлами разрешение коллизий при помощи цепочек работает быстрее, чем метод открытой адресации. Показано, что использование указанных методов хеширования позволяет достичь сопоставимых, а при увеличении размера хеш-таблицы – практически идентичных значений времени идентификации файловых объектов для микроядерных ОС в сравнении с монолитными системами. Это позволяет авторам сделать вывод о нецелесообразности применения подхода, реализованного в протоколе PUFFS.

Анализ работы всего стека ввода–вывода микроядерной ОС MINIX 3 с учетом дисковых операций показал, что при ограничении максимального числа хранимых структур *istruct* значениями до 1024 добавление хеш-таблиц не привносит заметного увеличения производительности. Это связано с тем, что медленные дисковые операции значительно сильнее влияют на производительность стека ввода-вывода, чем вычислительные операции внутри файловой системы. Однако с ростом числа хранимых в памяти *istruct*, вызванным как увеличением максимального числа открытых файлов, так и кэшированием самих структур *istruct* на случай повторного обращения к недавно закрытым файлам, использование хеш-таблицы становится необходимым.

Реализация ext2 для MINIX 3 выполнена при поддержке свободного университета Амстердама (VU University Amsterdam) и входит в состав операционной системы MINIX, начиная с версии 3.1.8. Авторы выражают признательность своему коллеге, научному программисту университета VU T. Veerman за плодотворное сотрудничество по тематике работы и полезные замечания.

Литература

1. Титов А.В. Методика оценки надежности встроенных программных средств при редких отказах // Изв. вузов. Приборостроение. – 2010. – Т. 53. – № 10. – С. 38–41.
2. Ostrand T.J., Weyuker E.J. The distribution of faults in a large industrial software system // SIGSOFT Softw. Eng. Notes. – 2002. – V. 27. – № 4. – P. 55−64.
3. Ostrand T.J., Weyuker E.J., Bell R.M. Where the bugs are // SIGSOFT Softw. Eng. Notes. – 2004. – V. 29. – № 4. – P. 86–96.
4. Chou A., Yang J., Chelf B. et al. An empirical study of operating systems errors // SIGOPS Oper. Syst. Rev. – 2001. – V. 35. – № 5. – P. 73–88.
5. Herder J.N., Bos H., Gras B. et al. Construction of a Highly Dependable Operating System // Proceedings of the Sixth European Dependable Computing Conference. EDCC '06. Washington, DC, USA: IEEE Computer Society. – 2006. – P. 3–12.
6. Leslie B., Chubb P., Fitzroy-dale N. et al. User-level Device Drivers: Achieved Performance // Journal of Computer Science and Technology. – 2005. – V. 20. – P. 654–664.
7. Kantee A. puffs – Pass-to-Userspace Framework File System // Proceedings of the AsiaBSDCon. – 2007. – P. 29–42.
8. Лав Р. Ядро Linux. Описание процесса разработки: Пер. с англ. – 3-е изд. − М.: Вильямс, 2013. – 496 с.
9. The VFS-FS protocol. MINIX 3 Developers Guide [Электронный ресурс]. – Режим доступа: http://wiki.minix3.org/en/DevelopersGuide/VfsFsProtocol, своб. Яз. англ. (дата обращения: 02.03.2013).
10. Traeger A., Zadok E., Joukov N., Wright C.P. A nine year study of file system and storage benchmarking // ACM Transactions on Storage (TOS). – 2008. – V. 4. – № 2. – P. 1–56.
11. Седжвик Р. Фундаментальные алгоритмы на C++. Части 1–4: Анализ/Структуры данных/Сортировка/Поиск: Пер. с англ. – Киев: Издательство «ДиаСофт», 2001. – 688 с.
12. Appleby A. Murmurhash v3 [Электронный ресурс]. – Режим доступа: http://sites.google.com/site/murmurhash/, свободный. Яз. англ. (дата обращения 16.03.2013).
13. Кнут Д.Э. Искусство программирования. Т. 3. Сортировка и поиск: Пер. с англ. – 2-е изд. – М.: Вильямс, 2000. – 832 с.

Иванов Евгений Юрьевич – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, студент i@eivanov.com

Косяков Михаил Сергеевич – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, кандидат технических наук, доцент, mkosyakov@gmail.com