УДК 004.451.87 + 004.451.88

ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ МИКРОЯДЕРНЫХ ОС

Дмитрук С.С., Теплинский С.В.
Донецкий национальный технический университет
кафедра Компьютерной инженерии

Аннотация

Дмитрук С.С., Теплинский С.В. Оптимизация производительсноит микроядерных ОС. *Рассматриваются основные факторы, приводящие к низкой производительности систем на основе микроядра. Выполняется анализ причин и способов уменьшения их влияния. Приводятся списки уже реализованных в исследовательской микроядерной ОС и запланированных к реализации в будущем способов повышения производительности на основе выполненного анализа.*

Введение

Одним из главных факторов, сдерживающих распространение микроядерных ОС (сдерживающих настолько, что современных систем такого типа крайне мало), является низкая производительность подобных систем, обусловленная особенностями их архитектуры. Данные особенности вводят такие ограничения на процессы взаимодействия между компонентами системы, что использование способов взаимодействия, характерных для монолитных систем, является неэффективным. И, следовательно, для повышения производительности микроядерных ОС необходимо проводить исследование особенностей обмена данными именно в этих системах с последующей разработкой специализированных методов обмена данными, а также специальных средств оптимизации.

Целью данной статьи является выявление и анализ главных причин более низкой производительности микроядерных систем по сравнению с другими видами архитектур ОС, а также описание возможных методов повышения производительности.

Обзор системы

В данной работе особенности микроядерных систем с точки зрения производительности рассматриваются на примере разработанной исследовательской операционной системы. В состав системы входят следующие компоненты:

ядро, включающее в себя также:
- встроенный в ядро (ограниченный) отладчик;
- нечто, подобное HAL (Hardware Abstraction Layer);

6 основных серверов:
- сервер поиска устройств;
- сервер поиска драйверов для устройств;
- сервер работы с файловыми системами;
- сервер инициализации;
- сервер управления памятью;
- сервер управления процессами;

6 серверов драйверов (devfs, fatgen, idedisk, mterminal, ramdisk, ramimg);

6 простых программ (date, echo, envlst, hwinfo, login, shell);

2 клиентские библиотеки:
- часть стандартной библиотеки языка программирования C;
- небольшая C++ библиотека поддержки (базовые коллекции, примитивы

синхронизации и т.д.).

Как видно, помимо самого микроядра имеется ряд клиентских и серверных процессов, разделённых на несколько функциональных групп. Группа из шести основных серверов выполняет часть функций, которые в случае монолитных систем включаются в состав ядра. Они выполняют высокоуровневые функции, часть из которых никак не представлена в микроядре. Следующая группа включает в себя реализации нескольких типов драйверов, которые обеспечивают базовые функции ввода/вывода, необходимые для нормального функционирования системы. Шесть программ-примеров являются программным обеспечением, с которым взаимодействует пользователь. А две библиотеки представляют собой вспомогательные средства для разработки.

Из списка выше видно, что многое из того, что принято считать частью ядра, вынесено из него в отдельные модули. Само же ядро имеет довольно скромный размер и выполняет несколько небольших групп функций:

общие функции, доступные всем:
- поиска процессов по именам;
- асинхронного обмена сообщениями;
- ускоренного обмена большими объёмами данных;
- учёта времени;
- чтения информации о процессе;

функции, доступные только драйверам:
- для работы с прерываниями;
- для работы с физической памятью;

функции, доступные только серверу памяти:
- выделения/освобождения страниц физической памяти;

функции, доступные только серверу процессов:
- создания/ветвления/завершения процессов;
- создания/приостановки/возобновления/завершения нитей;
- записи информации о процессе.

Узкие места с точки зрения производительности

Из приведённого выше списка функций видно, что ядро системы оперирует минимальным набором абстракций, ограничиваясь при этом довольно примитивными операциями. Но, несмотря, на немногочисленность и самодостаточность элементов, входящих в данный набор, они всё же требуют репликации во внешних по отношению к ядру модулях. При этом возникают накладные расходы не только на процессы копирования и обмена данными, но и на синхронизацию структур данных ядра с аналогичными или связанными структурами, хранящимися в основных серверах.

Ситуация усугубляется также тем, что в отличие от монолитных систем, в которых всем частям ядра предоставляется одинаковый уровень доверия, в микроядерных ОС всё, что находится вне микроядра, считается не доверенным настолько, насколько это возможно. Низкий уровень доверия означает ограничение доступа к системным данным, наличие дополнительных проверок, как корректности данных, так и аутентичности их источника, а также хранение в ядре избыточной при нормальной работе системы информации, которая призвана обеспечить безопасное продолжение работы в случае отказов в работе основных серверов.

Всё вышеперечисленное описывает только взаимодействие ядра с модулями и наоборот, и хотя потребляет часть ресурсов (возможно, даже значительную, при серьёзных ошибках в реализации) не является главной причиной более низкой производительности микроядерных ОС в сравнении с монолитными системами. Последнее в значительной

степени обусловлено сложностями, связанными либо с передачей данных между процессами, либо со значительными накладными расходами на частое переключение контекстов исполнения. Описание двух данных проблем приведено в подразделах ниже.

Обмен данными

В любой системе обмен данными между её компонентами играет очень важную роль, так как без него невозможно задать параметры задачи и получить результаты её решения. В операционных системах процессы обмена данными можно разделить по участникам на обмен между ядром и процессами и на обмен между двумя процессами. А по объёму передаваемых данных условно можно выделить крайне малые (десятки байт), малые (сотни байт), средние (до нескольких кибибайт) и относительно большие объёмы (несколько кибибайт и больше).

Так как ядро присутствует в адресном пространстве каждого из запущенных процессов, то доступ к данным процесса при операциях обмена типа ядро-процесс реализуется непосредственным копированием участка памяти (за исключением крайне малых объёмов, которые передаются через регистры). В случае обмена данными типа процесс-процесс при небольших объёмах ядро выполняет функции буферирования (это необходимо из-за асинхронного характера обмена сообщениями) и полностью копирует сообщения во внутренние структуры данных при отправке. Но обмен данными большого объёма не может осуществляться аналогичным образом по следующим причинам:

обмен сравнительно большими объёмами данных в операционной системе встречается достаточно часто. Это не только операции ввода/вывода с носителями, но и практически любые операции обмена, в которых буферирование данных может повысить производительность путём упаковки множества мелких запросов в один, но, возможно значительно больший по размеру;

процессы, участвующие в одном цикле обмена данных, выстраиваются в цепочку, что приводит к ощутимому росту накладных расходов при увеличении длины цепочки хотя бы на один элемент;

такой подход требует минимум двух дополнительных операций копирования (в ядро и из него), что приведёт к большим временным затратам.

Это приводит к необходимости применения альтернативного подхода к обмену большими объёмами данных. При этом асинхронность вносит свои ограничения на возможные решения данной задачи, так как период актуальности данных в буфере сообщения становится не детерминированным. Один из возможных способов эффективного обмена данными большого объёма приведён в разделе 3 данной работы.

Переключение контекстов

Под переключением контекстов понимается смена среды выполнения, которая включает в себя две главных операции [2]:

смену значений регистров (со сменой стеков);

смену адресного пространства (эта операция выполняется только при смене активного процесса).

При этом первая из них является относительно легковесной (хотя и эту операцию оптимизируют, отказываясь от её аппаратной реализации в пользу чисто программной). А вторая - более тяжёлой, прежде всего, из-за того, что приводит к сбросу TLB (Translation Look-aside Buffer) кеша процессора, который хранит соответствие виртуальных адресов памяти физическим адресам [1]. Частое переключение приводит к постоянной (хотя не обязательно полной) очистке кеша и, следовательно, все последующие операции с виртуальной памятью будут сопровождаться кеш-промахами, и, соответственно, эффективность кеширования адресов будет крайне низкой.

Для повышения эффективность использования TLB кеша, необходимо минимизировать количество переключений. Несколько примеров возможных оптимизаций приведены в следующих двух разделах.

Реализованные методы повышения производительности

Обмен большими объёмами данных на основе прав

С целью оптимизации передачи данных была введена дополнительная абстракция прав на доступ к данным, находящимся в адресном пространстве другого процесса. Кратко права можно описать следующим образом:

право выдаётся либо на запись, либо на чтение;

любой байт, на который выдано право доступа, может быть считанным или записан только один раз;

правом может воспользоваться только процесс, которому оно выдано;

текущий владелец может передать право любому процессу, при этом лишив себя этого права;

право аннулируется ядром при первой же возможности.

Сама же экономия достигается благодаря передаче между процессами не самих данных, а ссылки на право их использования. То есть, процесс создаёт право, оно передаётся через цепочку промежуточных процессов и реализуется конечным процессом-получателем. По завершению операций система удаляет право, что происходит без участия процесса, который выдал его.

Хранение части информации о процессе непосредственно в ядре

Так как получение доступа к данным другого процесса требует минимум двух переключений контекстов, то имеет смысл найти способ избежать этих переключений. Одним из таких способов является помещение информации в специальные области ядра, в которые часть из процессов имеют право записи. В этом случае обращение к (преимущественно) статическим данным может быть проведено без дополнительных издержек путём чтения данных из ядра через соответствующий интерфейс.

Кеширование и буферирование на различных уровнях

Кеширование данных с целью уменьшения количества системных вызовов, обращений к медленным накопителям, сетевых запросов и так далее играет огромную роль и в монолитных системах. Отключение подсистем кеширования ввода/вывода драйверов так и буферирования в пользовательском пространстве приведёт к резкому падению производительности любой системы, так как неизбежно приведёт к нерациональному использованию её ресурсов. В случае же микроядерных систем важность подобных средств повышения производительности только растёт, так как они позволяют откладывать и значительно уменьшать количество обменов данными и переключений между процессами, то есть минимизировать использование двух наиболее медленных процессов в микроядрах.

Предлагаемые методы повышения производительности

Особые приоритеты для основных процессов

Так как основные процессы выполняют наиболее важные функции ОС после микроядра, то прерывание их работы в большинстве случаев не имеет большого смысла. Во-первых, запросы к таким модулям не должны выполняться долго в виду нетребовательности выполняемых ими функций к ресурсам. Во-вторых, их прерывание скорее отрицательно скажется на эффективности работы всей системы, так как необработанные запросы к таким

модулям будут блокировать работу их клиентских приложений, которых в общем случае может быть довольно много. Исходя из описанных причин, становится очевидным, что система приоритетов должна быть откорректирована таким образом, чтобы основные модули системы монопольно владели процессорным ресурсом в течение необходимого им промежутка времени. То есть между основными модулями должна быть реализована кооперативная (не вытесняющая) многозадачность.

Особое планирование выделения квантов потокам, основанное на цепочках обмена данными

Обмен сообщениями при выполнении в системе различных операций можно описать передачами сообщений между последовательностями серверов. Путём анализа данной информации возможно осуществление оптимального планирования исполнения процессов. Например, если ядро выявило цепочку обмена A->B->C->D, то с большой долей вероятности можно сказать, что по завершению обработки запроса процессом D он инициирует обратный обмен сообщениями, то есть D->C->B->A. Обладая данной информацией, микроядро может выполнить сортировку процессов в очереди на выполнение (или снабдить очередь метаданными на будущее). Также может выполняться сбор статистики для оптимизации цепочек запросов, а не только ответов.

Хранение большего количества системной информации непосредственно в ядре

На данный момент система предоставляет возможность хранения только информации привязанной к процессам, и доступом на запись обладает только сервер управления процессами. Возможно, обобщение данного подхода с хранением части данных в ядре для обеспечения быстрого доступа, позволит оптимизировать работу с другими серверами, а не только с сервером менеджера процессов.

Выводы

Из работы видно, что причиной низкой производительности является не столько сама архитектура, сколько попытки применения в ней методов, не предназначенных и не подходящих для систем микроядерного типа. А для увеличения характеристик производительности системы необходима разработка новых, специализированных, методов, учитывающих процессы обмена, проходящие в микроядерных ОС.

Список литературы

1. Таненбаум Э., Вудхалл А., Операционные системы: разработка и реализация – СПб.: Питер, 2006. – 576 с.
2. Таненбаум Э., Современные операционные системы. 3-у изд. – СПб.: Питер, 2011. – 1120 с.