Секция «Вычислительная математика и кибернетика»

Увеличение межпроцессного взаимодействия в Microsoft Sngularity

Османов Гаджи Шамильевич
Школьник
НЦ ЛНМО, Лицей № 572, , Санкт-Петербург, Россия
E-mail: osmanovx@gmail.com

Операционные системы занимают очень важное место на рынке программного обеспечения. Невозможно представить себе не то что компьютер, но даже современной сотовый телефон без полноценной ОС внутри, сравнимой по функциональности с системами для полноценных компьютеров. Однако, прогресс в современных операционных системах очень медленный, и большинство популярных ОС (включая ОС для столь новых продуктов, как смартфоны и коммуникаторы) имеют внутреннюю архитектуру, отвечающую идеологии едва ли не 40-летней давности. С одной стороны, такая консервативность - показатель успеха традиционного подхода как с финансовой, так и с практической точек зрения.

С другой стороны, за 40 лет произошло множество изменений, и ОС, и ПО уже куда менее надежны и безопасны, чем хотелось бы большинству пользователей (и разработчиков!), и главное, чем возможно достигнуть с современным уровнем развития технологий в целом.

Среда вычислений тех времен, когда закладывались архитектурные основы современных ОС, крайне отличалась от современной. Компьютеры были весьма ограничены в скорости и объеме памяти; они использовались только малыми группами технически грамотных и не злонамеренных пользователей; они редко объединялись в сети или общались с физическими устройствами. Сейчас все не так, но современные архитектуры компьютеров, операционные системы и языки программирования недостаточно изменились для того, чтобы отражать фундаментальные изменения в компьютерах и их использовании. Именно поэтому многие фирмы и исследователи разрабатывают новые ОС на новых принципах. Одной из таких ОС является

Singularity, разрабатываемая компанией Майкрософт.

Singularity - это экспериментальная операционная система, разрабатываемая как основа для более надежного системного и прикладного ПО. Ключевой аспект Singularity - модель расширения, построенная на программно-изолированных процессах(Software-Isolated Process, SIP), которые инкапсулируют части приложения или системы и обеспечивают сокрытие информации, изоляцию сбоев и строго типизированный интерфейс. Связь между SIP осуществляется через двунаправленные, строго типизированные высокоуровневые каналы.

К сожалению, вместе с безопасностью эта идея несет и значительные издержки с точки зрения производительности, поскольку общение между процессами происходит через посредника (Exchange Heap). Причем потеря производительности весьма ощутима - скажем, на работе с файловой системой замедление может составить десятки раз.

Естественен вопрос, как повысить быстродействие связи между процессами, при этом, не потерять безопасность! Для этого в данном проекте предложена идея доработки ядра ОС Singularity, названная "технологией CSM(Contract Shared Memory")

CSM по сути является безопасной общей памятью. Безопасность осуществляется за счет контракта, по которому создаётся канал между процессами, а по каналу в свою очередь - передаются уникальные идентификаторы. Благодаря уникальным идентификаторам процесс - сервер даёт процессу - клиенту доступ к созданной процессом - сервером общей памяти.

За счет контракта, мы не теряем безопасность, а благодаря прямому доступу процесса к общей памяти - решается проблема с быстродействием. Иначе говоря, если между процессами не создан канал, то они не могут использовать общую память.

Реализация CSM была разбита на 9 модулей:

1) Создание в Singularity объект CSM, с указанным размером и с уникальным идентификатором

При создании общей памяти мы указываем размер общей памяти, а также уникальный идентификатор, по которому мы в дальнейшем будем получать доступ для работы (запись или чтение данных) с общей памятью.

2) Реализация основных функций - записи и чтение данных в CSM

Все данные имеют уникальные имена, по которым в будущем и будет происходить обращение к данным.

3) Создание проверок на совместимость вызываемых или записываемых данных.

При записи данных для безопасности происходит контроль типов, необходимый для безопасности ОС.

4) Инициализация объекта в Singularity

5) Создание безопасного контракта, по которому будет создаваться канал между процессами

Благодаря каналу, задаваемым контрактом, мы реализуем безопасность CSM, поэтому мы прежде чем создать канал, должны создать контракт, который содержит алгоритм отправки универсальных идентификаторов общей памяти. Пример работы алгоритма следующий.

А) Процесс – сервер создаёт общую память по универсальному идентификатору

Б) Данный идентификатор отправляется процессу – клиенту

В) После получения идентификатора процесс – клиент отправляет подтверждение о полученном идентификаторе

Г) Процесс - сервер инициализирует общую память, тем самым даёт доступ к чтению и записи данных в общую память.

6) Создание новых API(в Singularity ABI) функций:

Share - данная функция записывает в общую память данные

DataOut - данная функция читает данные из общей памяти

SizeOut - данная функция возвращает размер общей памяти

7) Создание Сервиса CSM, благодаря которому процессы смогут обращаться к Singularity и вызывать уже ABI функции CSM.

Просто обратиться напрямую к ядру невозможно, поэтому сперва нужно создать сервис, который проверяет правильность вводимых аргументов для ABI функций.

8) Реализация функции Back In Time. Данная функция восстанавливает данные во времени. Она содержит историю изменений данных, а также записывает ненужные процессу данные в виртуальную память, для того, чтобы не заполнять оперативную!

Данная функция - необязательна, разработчик сам может выбирать использовать ее или нет!

9) Написание тестов, тестирующих быстродействие работы CSM.

Разумеется для реализации данных модулей пришлось значительно изменять ядро для инициализации технологии CSM.

Последовательность работы CSM следующая:

1)Процесс (сервер) подключает сервис CSM и создаёт канал с процессами - клиентами

2)Процесс (сервер) вызывает ABI функцию

3)Создаётся общая память, для доступа к ней используется уникальный идентификатор, задаваемый процессом (сервером)

4)По каналу уникальный идентификатор передаётся другим процессам

5)Процессы получают уникальный идентификатор и обращаются к общей памяти по нему.

В результате тестирования технологии CSM, работа связи между процессами показала значительный прирост в производительности, до 10, а то и в 10 раз! в зависимости от размера передаваемых данных.

Тест 1. Создаётся два процесса, которые передают друг другу 500 однозначных чисел, через канал, с использованием Exchanged Heap, а также с использованием csm. Один процесс выводит в консоль “Starting original test”, после этого он посылает 250 чисел, другой их принимает, потом принимающий процесс начинает отправлять 250 чисел процессу изначально посылающий сообщения, после этого процесс пишет в консоль “End original Test” Засекается время выполнения с появления 1 надписи, а заканчивается с появлением второй надписи.

Тест 1 Время исполнения (сек)

Channel 456

CSM 64

Тест 2. Создаётся два процесса, которые передают друг другу два массива с элементами(500 однозначными числами), через канал, с использованием Exchanged Heap, а также с использованием csm. Один процесс выводит в консоль “Starting original test”, после этого он посылает массив с 250 числами, другой их принимает, потом принимающий процесс начинает отправлять массив с 250 числами процессу изначально посылающий сообщения, после этого процесс пишет в консоль “End original Test” Засекается время выполнения с появления 1 надписи, а заканчивается с поянием второй надписи

Тест 2 Время исполнения (сек)

Channel 745

CSM 158

Помимо данных тестов были реализованы и другие

Тест 3 заключается в измерении времени чтения и записи в файл. Тест был сделан в 2-х версиях. Одна версия с использованием стандартной технологии, другая с использованием технологии CSM. Для реализации данного теста, пришлось внедрять свою технологию в дисковый драйвер.

Тест 3 Время исполнения (сек)

Channel 841

CSM 164

Тест 4 заключается в имерении времени обрабатывания клиентских запросов в секунду (веб-сервер). Тест был сделан в 2-х версиях. Одна версия с использованием стандартной технологии, другая с использованием технологии CSM

Для реализации данного теста, пришлось внедрять свою технологию в дисковый драйвер, в сетевой драйвер, а также встраивать в сам веб-сервер, поскольку он состоит из нескольких процессов!

Тест 4 Время исполнения (сек)
Channel 847
CSM 173

Литература

1. http://www.rsdn.ru/article/singularity/singularity.xml (обзор Singularity)

2. http://singularity.codeplex.com/ (официальный сайт)

3. http://research.microsoft.com/pubs/69431/osr2007_rethinkingsoftwarestack.pdf (Singularity: Rethinking the Software Stack)

4. http://research.microsoft.com/en-us/projects/singularity/ (документация по самой ОС)

5. http://research.microsoft.com/en-us/projects/singularity/asplos2008_singularity_rdk_tutoria (Using the Singularity Research Development Kit)

6. Language Support for Fast and Reliable Message-based Communication in Singularity OS. Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen C. Hunt, James R. Larus, and Steven Levi. Proceedings of EuroSys2006. Leuven, Belgium, April 2006. ACM SIGOPS.

7. An Overview of the Singularity Project, Technical Report MSR-TR-2005-135, Microsoft Research, 2005.

8. Edmund B. Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt, Helios: Heterogeneous Multiprocessing with Satellite Kernels, in Proceedings of the 22nd Symposium on Operating Systems Principles (SOSP '09), Association for Computing Machinery, Inc., Big Sky, MT, October 2009

9. Galen C. Hunt and James R. Larus, Singularity: Rethinking the Software Stack, in ACM SIGOPS Operating Systems Review, vol. 41, no. 2, pp. 37-49, Association for Computing Machinery, Inc., April 2007