ВЗАИМОДЕЙСТВИЕ МОДУЛЕЙ ЯДРА ОПЕРАЦИОННОЙ СИСТЕМЫ С МИКРОЯДЕРНОЙ АРХИТЕКТУРОЙ

Шлык П.А.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Оношко Д.Е. – ст. преподаватель

В работе рассматриваются основные принципы и концепции взаимодействия модулей ядра, используемые в процессе разработки прототипа операционной системы для платформы IA-32.

Основным недостатком операционных систем с микроядерной архитектурой является меньшая производительность, если сравнивать с операционными системами с другими архитектурами. Производительность операционной системы снижается по следующим причинам [1]:

1) высокая стоимость перехода из режима пользователя в режим ядра;

2) высокая стоимость переключения контекста исполнения задачи;

3) неэффективная иерархия взаимодействия между модулями ядра.

Дополнительной проблемой коммуникации модулей является возможность возникновения ситуации циклической зависимости данных между модулями. Ядро должно предлагать структурные гарантии по недопущении циклической зависимости. Выделяются несколько способов:

1) однонаправленный порядок сообщений (partial message ordering);

2) системные политики управления сообщениями.

Однонаправленный порядок сообщений предполагает ограничения на отправку запросов-ответов клиента-сервера. Например, если процесс A отправил сообщение процессу B, то процесс B не может отправить сообщение процессу A.

Системные политики позволяют конфигурировать те процессы, которым заданный процесс модуля ядра может отправлять или от которых процесс может получать сообщения.

Выделяется несколько типов взаимодействия в операционной системе с микроядерной архитектурой:

1) модуль-ядро;

2) модуль-модуль;

3) ядро-модуль.

Тип взаимодействия модуль-модуль, несмотря на кажущийся прирост производительности за счёт уменьшения количества обращений к ядру, приводит к усложнению топологии взаимодействия между модулями системы и, как следствие, к уменьшению надёжности системы в целом. Потому в процессе разработки прототипа операционной системы, было принято решение заменить взаимодействие между модулями ядра на взаимодействие модуля с самостоятельной подсистемой ядра. Тем самым достигается:

1) контроль за однонаправленным порядком сообщений;

2) отсутствие зависимостей модулей между собой и, как следствие, большая изолированность модулей между собой.

Хотя с точки зрения пользователя, ядро по-прежнему предоставляет единую функциональность, внутренняя структура ядра становится более модульной и устойчивой к отказам.

В качестве примера обработки пользовательского запроса предлагается выполнение операции записи над файлом (см. рисунок 1).

Запрос пользователя преобразуется в запрос виртуальной файловой системы (VFS) к соответствующему модулю файловой системы [2]. Модуль файловой системы выполняет запрос к подсистеме управления периферийными устройствами (IO). Подсистема IO преобразует запрос от модуля файловой системы в запрос к модулю соответствующего блочного устройства.

Ключевым элементом в реализации механизмов коммуникации в операционной системе являются каналы передачи сообщений. Выделяются следующие типы каналов:

1) каналы встречной пересылки сообщений (rendezvous messaging);

2) каналы буферизированной отправки сообщений.

Каналы встречной пересылки сообщений предоставляют синхронный способ для коммуникации между модулями: отправитель (или получатель) блокируется до тех пор, пока соответствующий получатель (отправитель) не будет готов для обработки сообщения. Встречная пересылка сообщений проста в реализации, но обладает рядом существенных недостатков:

1) потенциальная блокировка задач ядра;

2) задержка на ожидание действий от встречной стороны и, как следствие, снижение производительности модулей ядра;

3) вероятность возникновения состояния гонки (например, при обработке вложенных прерываний).

Каналы буферизированной отправки сообщений позволяют реализовать асинхронную модель взаимодействия между модуля ядра. Но, с другой стороны, затрудняется отладка модели коммуникации между модулями и увеличивается сложность всей системы:

1) проблема блокировки не решается полностью, так как задача может быть заблокирована при заполнении буфера сообщений;

2) дополнительные накладные расходы на поддержание очереди сообщений;

3) потенциальное увеличение времени отклика модулей системы на внешние события за счёт буферизации сообщений.

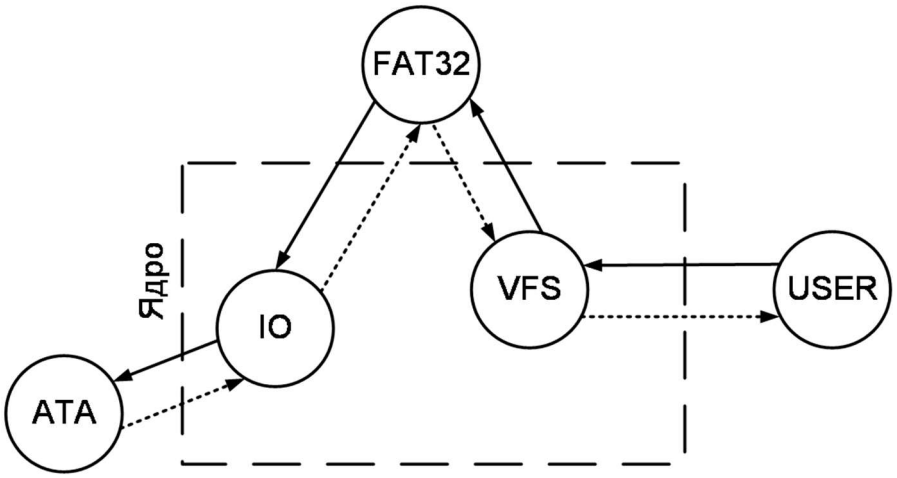

Рисунок 1 – Модель взаимодействия модулей ядра на примере обработки пользовательского запроса на запись в файл

Было принято решение использовать каналы буферизированной отправки сообщений в качестве основного средства коммуникации компонентов операционной системы.

Некоторые операционные системы производят периодическое сканирование каналов передачи сообщения на предмет блокировки между отправителем и получателем [3]. Это позволяет выявлять и разрешать состояния взаимной блокировки (deadlock), но требует дополнительных вычислительных ресурсов и усложняет управление системой. Кроме того, такой подход не всегда гарантирует своевременное обнаружение проблем, потому в разработанном прототипе операционной системы данный подход не используется.

Ядро операционной системы должно предоставить модулями периферийных устройств специализированный набор системных вызовов для реализации следующей функциональности:

1) конфигурация обработчика прерываний;

2) доступ к пространству портов ввода-вывода;

3) доступ к физической памяти.

Для платформы IA-32 обработчик прерываний представляет собой последовательность инструкций записи в память или в порты. Следовательно, конфигурация обработчика прерывания представляет собой задание этой последовательности инструкций. Но более детальное изучение вопроса позволит ограничить последовательность инструкций до одной: записи в определённый регистр периферийного устройства. И конфигурацией обработчика прерывания будет выбор одного из стандартных обработчиков, реализованного в ядре.

Для работы с устройствами, использующими прямой доступ к памяти (DMA) или регистры, расположенные в физической памяти, драйверы должны иметь возможность отображать физическую память в свое виртуальное адресное пространство. Следовательно, ядро должно предоставить системный вызов для безопасного отображения физической памяти, проверки прав и управления страницами памяти.

Можно сделать вывод, что операционная система с микроядерной архитектурой должна поддерживать как структурные возможности для контроля порядка взаимодействия между модулями ядра, так и средства коммуникации, подходящие под требования предметной области.

Список использованных источников:

1. Танненбаум, Э. Современные операционные системы / Э. Танненбаум, Х. Бос – 4-е изд. –СПб.: Питер, 2015. 1120 с.

2. Mauerer, W. Professional Linux Kernel Architecture / Wolfgang Mauerer – John Wiley & Sons, 2010. 1368 p.

3. Herder, Jorrit N. Towards a true microkernel operating system / Jorrit N. Herder. – Amsterdam:Vrije Universiteit, 2005. 117 p