Санкт-Петербургский государственный университет

Кафедра системного программирования

Чусовитин Денис Андреевич

Управление адресными пространствами для встроенных устройств

Дипломная работа

Научный руководитель:
ассистент Козлов А. П.

Рецензент:
программист ООО "Вайс-Техника" Дерюгин Д. Е.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Department of Software Engineering

Chusovitin Denis

Address space management for embedded devices

Graduation Thesis

Scientific supervisor:
assistant Anton Kozlov

Reviewer:
Software Developer at WiseTech Ltd Denis Deryugin

Saint-Petersburg
2017

Оглавление

Введение

Операционные системы (ОС) выполняют широкий диапазон задач, и, соответственно, сильно различаются по количеству доступных ресурсов. Одним из таких ресурсов является память. Устройства, на которых работают ОС, часто имеют подсистемы управления памятью, которые упрощают взаимодействие с процессором. Также, такие подсистемы предоставляют дополнительные функции, например защиту на запись или чтение в определенные блоки памяти. Это широко используется в ОС для защиты данных приложений и собственных компонент ядра. Кроме того, в операционных системах реального времени встает вопрос о скорости доступа к памяти. Это приводит к дополнительным требованиям к подсистемам управления памятью, из-за чего часто приходится отказываться от многих возможностей таких подсистем.

Для каждого конкретного класса устройств эффективен свой способ управлению памятью. Всего существует два вида подсистем: MMU и MPU.

MMU (Memory Management Unit) – устройство управлению памятью. Его основные функции заключаются в трансляции виртуального адресного пространства в физическое, защите памяти и управлению кэш-памятью. MMU в основном используется в устройствах, выполняющих прикладные задачи (например, в процессорах Cortex-A[1]).

MPU (Memory Protection Unit) – устройство защиты памяти. Из всех функций MMU обеспечивает только защиту памяти. Используется в специализированных устройствах, например в Cortex-M[2] (для встраиваемых систем) или Cortex-R[3] (для систем реального времени).

Данная работа выполнялась на базе ОС Embox[6] – активно разрабатываемой встроенной системы. Разработка ведется в тесном сотрудничестве с кафедрой системного программирования математико-механического факультета Санкт-Петербургского государственного университета. В данный момент Embox поддерживает ряд платформ (x86, MicroBlaze, ARM, PowerPC и другие). И что более важно, использует различные подсистемы управления памятью. Виртуальная память,

имеющаяся в ОС Embox на данный момент, пользуется в основном MMU, и имеет ряд ограничений, что не позволяет её применять в ряде случаев, например, использовать её на платформах с малым количеством оперативной памяти и в задачах, требующих высокой скорости отклика системы.

1. Постановка задачи

Целью данной дипломной работы являлась поддержка MPU в рамках проекта Embox и использование его для защиты различных частей ядра ОС и приложений. Для ее решения были сформулированы следующие задачи:

- спроектировать архитектуру подсистемы защиты памяти с использованием MPU в Cortex-M.
- реализовать драйвер MPU для Cortex-M.
- внести изменения в ОС Embox для поддержки защиты приложений и различных компонент ядра ОС.
- провести апробацию на плате STM32F4Discovery[8].

2. Обзор

2.1. MPU во встроенных системах

2.1.1. Общее описание MPU

Во встроенных системах MPU применяется для повышения их устойчивости и надежности с помощью:

- запретов пользовательским приложениям на доступ к данным, используемыми важными процессами (например, ядром операционной системы).
- запретов на исполнение кода из определенных участков памяти для предотвращения непроизвольного доступа и порчи данных.
- специальных атрибутов доступа к памяти.

2.1.2. Сравнение с MMU

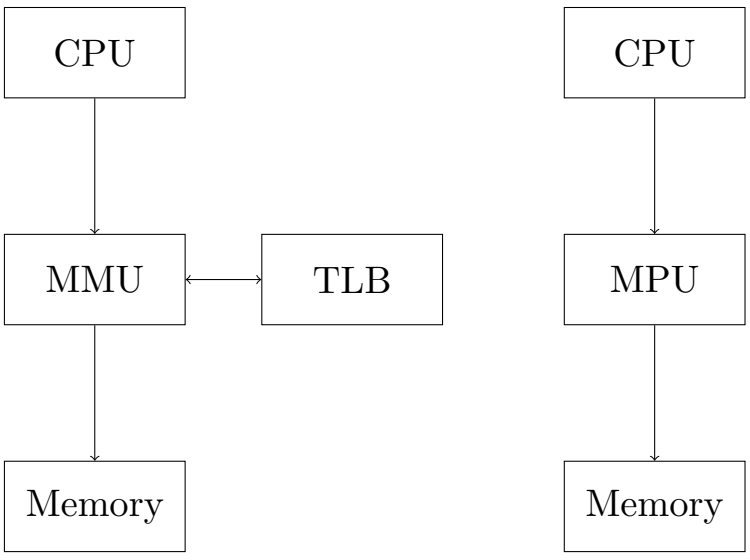

Рис. 1: Сравнение MMU и MPU

По сравнению с MMU, MPU не имеет буфера ассоциативной трансляции (TLB – Translation lookaside buffer), а также в нем нет таблицы страниц. (рис. 1)

Но в системах реального времени с ограничениями по скорости отклика использование MMU имеет серьезный недостаток: при трансляции из виртуального адресного пространства в физическое может произойти промах кэш-памяти (то есть транслируемый адрес может не оказаться в кэше). Это приводит к тому, что приходится считать физический адрес по таблице страниц. Таким образом, использование MMU может привести к непостоянному времени отклика.

В MPU все виртуальное адресное пространство сопоставляется к физическому как 1 к 1. По этой причине использование MPU популярно в системах реального времени.

2.1.3. Описание MPU в Cortex-M

MPU в Cortex-M может быть использован для защиты 8 регионов памяти. Каждый из них имеет свой номер (от 0 до 7), который соответствуют их приоритету. Таким образом, номера участка памяти в MPU – это их унифицированные идентификационные номера в системе. Также существуют специальные флаги доступа (на запись, чтение, исполнение и кэширование) которые можно выставлять регионам.

Участки памяти в MPU всегда содержит следующие атрибуты:

- номер участка в MPU.
- адрес участка памяти.
- флаги доступа к участку и его размер.

При разработке драйвера MPU следует учитывать, что участки памяти могут пересекаться между собой. Тогда уровень доступа к общему региону памяти определяется по участку с наибольшим приоритетом.

Также адрес участка памяти должен соотноситься со своим размером. Например, если размер составляет 64KB, то адрес должен быть кратен 64KB. Так как 64KB эквивалентно 0x10000, то адрес участка должен быть равен 0x10000, 0x20000 и так далее. Это необходимо для того, чтобы участок памяти был выровнен по своему размеру.

Флаги доступа и размер участков являются основными параметрами для системы, использующей MPU.

3. Драйвер MPU

3.1. Поддержка флагов доступа

Всего под атрибуты каждого участка памяти выделено 28 бит (рис. 2).

Биты	Имя флага	Описание
28	XN	Доступ на исполнение
26:24	AP	Флаги доступа к данным
21:19	TEX	Тип блока памяти
18	S	Разделяемая память
17	C	Возможность кэширования
16	B	Возможность буферизации
15:8	SRD	Флаги субрегионов
5:1	SIZE	Определяет размер участка

Рис. 2: Атрибуты блока памяти

Флаг XN контролирует исполнение кода. Для выполнения инструкции в пределах участка памяти, необходимо иметь доступ на чтение в привилегированном режиме и XN должен быть равен 0. В противном случае возникнет исключение.

Поле разрешения доступа (AP) определяет доступность участка памяти для чтения и записи в зависимости от режима (пользователя или ядра).

3.2. Обработка исключений

Исключение может быть вызвано из-за исполнения инструкций, вызывающие исключения, или вызвано как ответ на поведение системы, такое как прерывание, ошибка шины адреса, или вызвано умышленно при отладке[4].

Вход исключения происходит, когда ожидающее обработки исключение имеет достаточный приоритет. При этом процессор должен находится в режиме потока, то есть во время исполнения программное обеспечения пользователя или системы.

Если новое исключение имеет более высокий приоритет, чем то, которое сейчас обрабатывается, то новое исключение вытесняет старое. При вытеснении предыдущее исключение сохраняется на стеке. Его обработка продолжается после завершения нового.

Также исключение должно иметь достаточный приоритет для обработки. Это означает, что исключение имеет приоритет выше, чем все ограничения, выставленные специальными регистрами масок исключений. Исключения с приоритетом меньшем, чем текущий, не обрабатываются процессором.

Сразу после отправки на стек, указатель на него указывает на наименьший адрес в стековом кадре. Смещение кадра контролируется с помощью бита STKALIGN в регистре управления конфигурацией.

Стековый кадр содержит адрес возврата – адреса следующей инструкции в прерванной программе. Это значение восстанавливается на регистре PC во время возврата исключения, то есть там, где прерванная программа возобновляет работу.

Параллельно операциям на стеке выполняются чтение стартового адреса обработчика исключений из таблицы векторов прерываний. Когда операции на стеке завершаются, запускается выполнение обработчика. В тоже время на стек записывается значение в LR. Это указывает, какой указатель на стек соответствует стековому кадру, и в каком режиме находился процессор перед тем, как произошло исключение.

Если не возникло исключения с более высоким приоритетом во время входа, то вызывается обработчик исключения, при этом статус соответствующего ожидающего прерывания меняется на действующий.

Если во время входа пришло исключение с более высоким приоритетом, то для него запускается обработчик. Для предыдущего исключения ожидающий статус не меняется.

Возврат исключения происходит, когда процессор находится в режиме обработчика, и при этом выполняется одна из инструкции, которая загружает значение возврата на PC:

- загружается регистр PC.

- загружается значение на PC через инструкцию LDR.
- затем создается ветка с обработчиком исключения.

3.3. Переключение контекстов

Регистры MPU могу быть изменены только под привилегированным режимом, поэтому необходимо переключаться в данный режим перед внесением нужных изменений в регистрах. При этом нельзя переходить из пользовательского режима в непривилегированный напрямую, так как инструкции перехода доступны только в режиме ядра.

Эта проблема была решения с помощью исключений. В процессорах архитектуры Cortex-M 3/4 обработчик исключений находится всегда в привилегированном режиме, поэтому переключение можно сделать через вызов супервизора (SVC).

Для переключения контекста в системе использовались прерывания и стеки. Во время переключения, система должна сохранять состояние предыдущих задач, чтобы восстановить их при очередном переключении. При переключении контекста необходимо:

- сохранить состояние текущей задачи в таблице задач.
- обновить индекс текущей задачи на следующую, которую необходимо выполнить.
- установить процессор на использование нового стека.
- загрузить контекст задачи, которая будет исполняться.

3.4. Обновление флагов доступа

Для обновления атрибутов блока памяти необходимо, чтобы были выполнены следующие действия (рис. 3)[5]:

- Данный участок памяти должен быть отключен.

- Чтобы избежать неопределенного поведения, нужно отключить прерывание.
- Также перед обновлением MPU необходимо выполнить инструкции DSB и ISB. Это нужно для того, чтобы быть уверенными, что все незавершенные операции с памятью выполнены.
- Вышесказанные инструкции не требуются, если процедура обновления проходит в обработчике исключений, так как в этом случае завершение операций автоматически произойдет во время входа и выхода из исключения.

Рис. 3: Обновление флагов блока памяти

4. Защита приложений и различных частей ядра ОС

4.1. Защита от переполнения стека

Неограниченный рост стека может привести к сбоям в работе операционной системы, порче или даже потере данных пользователя. Такое может произойти, если запустить, например, бесконечную рекурсивную функцию, и тем самым привести к неизбежным проблемам в работе системы. Поэтому операционные системы должны уметь следить за состоянием стека, и завершать работу программы в случае его переполнения.

Для предотвращения вышеизложенных проблем необходимо вызывать исключение, как только произошло переполнение. Для этого на стеке выделяется специальный участок с защитой на запись. (рис. 4)

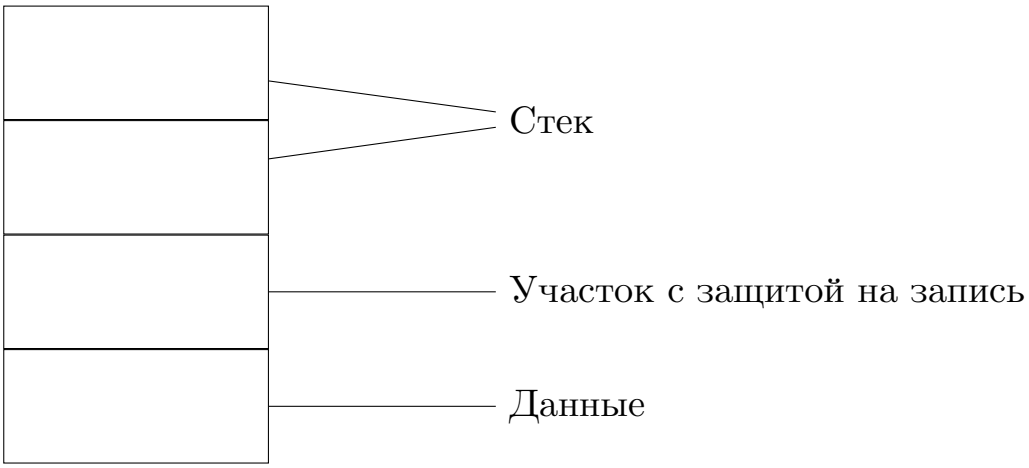

Рис. 4: Структура стека

Стек растет сверху вниз, и при переполнении происходит попытка записи в защищенную область памяти, что приводит к исключению, останавливающему работу программы. При этом данные остаются нетронутыми.

Для тестирования защиты от переполнения создается поток, запускающий бесконечную рекурсивную функцию.

4.2. Защита сегментов данных

Благодаря поддержке в MPU флагов доступа для чтения и записи можно реализовать защиту важных сегментов памяти. Например, с их помощью можно защищать данные приложений пользователя.

Для этого при переключения контекста участок с данными отмечается флагами с защитой на чтение и запись. В итоге, текущий поток не имеет доступа данным других потоков, только к своим.

В тестировании защиты сегментов данных запускаются два приложения, одно из которых пытаться испортить данные другого (рис. 5).

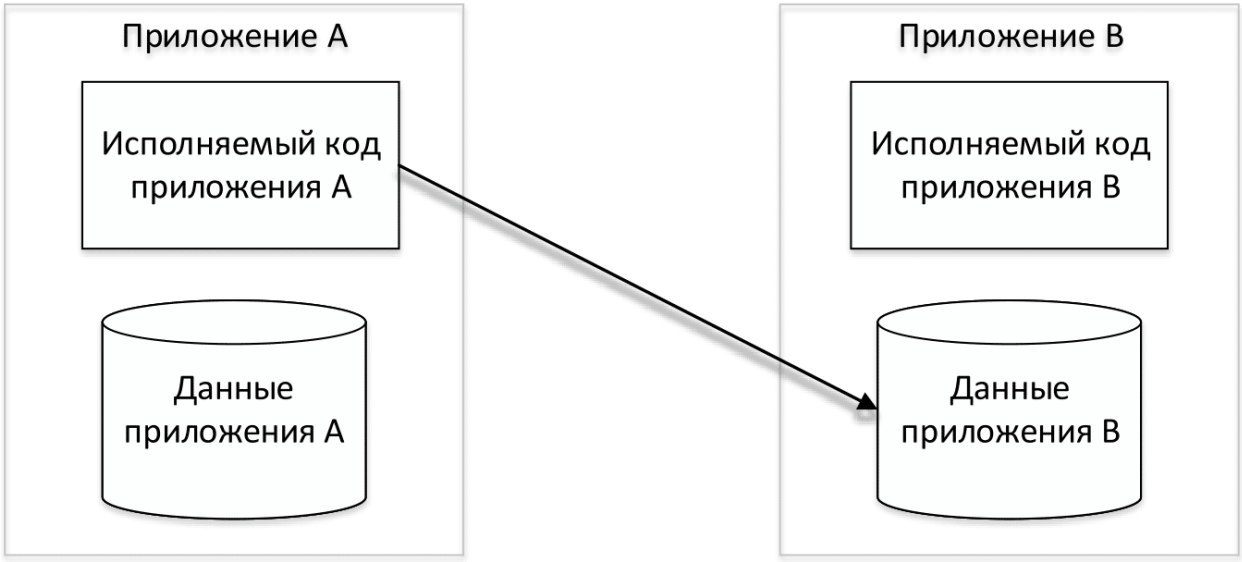

Рис. 5: Непроизвольный доступ к данным на примере двух приложений

4.3. Защита от исполнения

Возможность исполнения инструкций из любой области памяти может привести к порче или потерям данных. Кроме того, с помощью инъекций кода можно получить доступ даже к защищенным участкам памяти.

Но так как, кроме защиты на чтение и запись, MPU поддерживает защиту от исполнения, то для решения вышесказанных проблем достаточно выставить соответствующий флаг доступа.

Для тестирования защиты от исполнения был написан простой тест: скопировать функцию в массив из участка памяти с флагом NX и попытаться её выполнить.

5. Апробация на плате

Прототип драйвер был сделан на ARMv7 с использованием MMU на QEMU[7] - платформы для эмуляции аппаратного обеспечения. При этом единственными функциями MMU были:

- Отображение адресных пространств 1 к 1
- Использовать только флаги защиты страниц

То есть по сути MMU использовал как MPU. Такое решение было принято в связи с тем, что на QEMU проще отлаживать, чем на настоящей плате.

Для апробации на плате в систему регрессионного тестирования Embox'а были внедрены тесты, описанные в главе 4. Сами тесты были написаны на архитектуре ARMv7, затем перенесены на Cortex-M вместе с драйвером.

Заключение

В ходе выполнения данной дипломной работы были получены следующие результаты:

- спроектирована архитектура для драйвера MPU в Cortex-M.
- реализован драйвер MPU для Cortex-M. Сам драйвер был написан на язык ассемблера для соответствующей архитектуры, а интерфейс для работы с ним был создан на C.
- Были внести изменения в ОС Embox для поддержки защиты приложений и различных компонент ядра ОС.
- проведена апробация на плате STM32F4Discovery. Для тестирования были использованы тесты, описанные в главе 4.

Список литературы

[1] ARM. Cortex-A characteristics. — URL: http://www.arm.com/products/processors/cortex-a (дата обращения: 15.04.2017).

[2] ARM. Cortex-M Series Family. — URL: http://www.arm.com/products/processors/cortex-m (дата обращения: 16.04.2017).

[3] ARM. Cortex-R Series Family. — URL: http://www.arm.com/products/processors/cortex-r (дата обращения: 16.04.2017).

[4] ARM. Cortex-M4 Devices: Generic User Guide. — 2010.

[5] Atmel. Atmel AT02346: Using the MPU on Atmel Cortex-M3 / Cortex-M4 based Microcontrollers. — 2013.

[6] Github. Домашняя страница проекта Embox. — 2017. — URL: https://github.com/embox/embox (дата обращения: 08.04.2017).

[7] QEMU. Generic and open source machine emulator and virtualizer. — URL: http://www.qemu.org/ (дата обращения: 16.05.2017).

[8] STMicroelectronics. UM1472 User manual. — 2016.