Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б15-мм

Работа с таймерами процессора архитектуры mips64

Родионов Максим Александрович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ст. преподаватель кафедры ИАС, К. К. Смирнов

Консультант:
Ведущий разработчик ПО, ООО «СофтКом», И. С. Архипов

Санкт-Петербург
2024

Оглавление

Введение

В современном мире вычислительных систем системные таймеры играют ключевую роль в эффективном планировании задач, управлении ресурсами и поддержании стабильности системы. Они обеспечивают необходимую точность для выполнения критически важных функций, включая координацию процессов и обработку прерываний.

ООО «Софтком» разрабатывает операционную систему, предназначенную для создания многоагентных систем[1]. Важным аспектом такой системы является обеспечение предсказуемой (детерминированной) реакции на событие в определенном временном интервале. Для достижения этой цели используются аппаратные таймеры, которые позволяют отслеживать время выполнения задач и имеют более низкие накладные расходы, в сравнении с программными таймерами.

В работе предлагается реализовать взаимодействие с таймерами специализированного процессора архитектуры MIPS64. Основными компонентами каждого из таймеров являются аппаратный счетчик, который отсчитывает время от заданного значения; регистр управления, используемый для настройки и конфигурирования таймера; регистр состояния, который отображает текущее состояние таймера и предоставляет информацию о его работе. Реализация взаимодействия с этими компонентами позволит создать интерфейс, отвечающий требованиям операционной системы.

[1] https://github.com/IvanArkhipov1999/Martos

1. Постановка задачи

Целью работы является добавление возможности взаимодействия с аппаратными таймерами. Для её выполнения были поставлены следующие задачи:

1. провести обзор API[2] таймеров существующих операционных систем реального времени;
2. улучшить существующее в системе API таймеров;
3. спроектировать структуру блока таймеров;
4. реализовать функции из API.

[2]API (Application Programming Interface) — программный интерфейс приложений

2. Обзор

В современных операционных системах управление временем и синхронизация процессов играют ключевую роль в обеспечении их эффективной работы. Эти аспекты особенно важны для операционных систем реального времени, где точность таймера и возможность обработки событий в заданные сроки являются основополагающими элементами. В данном разделе будет представлен обзор API таймеров наиболее популярных операционных систем реального времени.

2.1. FreeRTOS

FreeRTOS [3] — лидирующая на рынке [2] операционная система реального времени с открытым исходным кодом, написанная на C. Она использует программные таймеры, которые не требуют аппаратной поддержки и не зависят от аппаратных таймеров.

В FreeRTOS функции обратного вызова программного таймера выполняются в контексте одной и той же задачи-демона, которая создаётся автоматически при запуске планировщика FreeRTOS. Приоритет и размер стека этой задачи задаются специальными константами. Функции API программного таймера отправляют команды из вызывающей задачи в задачу-демон через очередь, называемую «очередью команд таймера». Ключевым моментом является то, что таймер начинает отсчет времени с момента вызова соответствующей API-функции, а не с момента, когда команда была извлечена из очереди. Это достигается благодаря тому, что в очередь команд добавляется информация о значении счетчика системных квантов. Задача демона планируется так же, как и любая другая задача FreeRTOS, и будет обрабатывать команды или выполнять функции обратного вызова таймера только в том случае, если она имеет наивысший приоритет среди доступных задач. При этом время процессора не тратится на обслуживание таймеров во время их отсчета. Управление задачей обслуживания таймеров передается только при вызове API-функции для работы с таймерами или при срабатывании одного из таймеров.

В FreeRTOS предусмотрено два типа программных таймеров: одноразовые, которые выполняют свою функцию обратного вызова только один раз; и автоматические, которые перезапускаются по истечении установленного времени, обеспечивая периодическое выполнение функции обратного вызова. Обновлять режим таймера можно с помощью функции vTimerSetReloadMode().

Рассмотрим подробнее основные функции API таймера. Для создания таймера используется функция xTimerCreate(), в которой происходит инициализация полей структуры таймера входными параметрами. Программные таймеры создаются в неактивном состоянии. Для того чтобы перевести его в активное состояние можно воспользоваться одной из трех функций: xTimerStart(), xTimerReset() или xTimerChangePeriod(), которые запускают, перезапускают или изменяют период таймера соответственно. Получив параметры, эти функции вызывают xTimerGenericCommand(), внутри которой необходимая команда отправляется в очередь команд таймера. Аналогично работает функция xTimerStop(), которая переводит таймер в неактивное состояние.

2.2. ChibiOS

ChibiOS [1] — операционная система реального времени (RTOS), написанная на C. Она использует как программные, так и аппаратные таймеры. Аппаратные таймеры используются для управления системным временем, поэтому они инициализируются и запускаются во время инициализации всей системы.

Для начала определим понятие системного счетчика времени. Системный счетчик времени — это элемент, который увеличивается на единицу при каждом тике системы. В ChibiOS предусмотрены два режима обработки системного времени. Первый из них - Classic Mode (Классический режим), где системное время представлено как обычная переменная, которая увеличивается с помощью периодического прерывания, выполняемого с заданной частотой. Преимущество этого режима

заключается в простоте его реализации, однако недостатком является необходимость постоянной обработки прерываний центральным процессором для обновления счетчика. Второй режим - Tick-less Mode (Режим без тиков), в котором системное время определяется аппаратным счетчиком, обновление которого не требует прерываний. Важно отметить, что режим работы с системным временем скрыт в API, и для приложения оба режима выглядят одинаково.

В API для работы с системным временем доступны следующие функции: chVTGetSystemTime(), которая возвращает текущее системное время; chVTTimeElapsedSinceX(), которая возвращает интервал, прошедший с указанного системного времени; chVTIsSystemTimeWithin(), которая возвращает значение true, если текущее системное время находится в пределах указанного периода. Рассмотрим каждую из этих функций в режиме без тиков. Функция chVTGetSystemTime обращается к аппаратному таймеру для получения текущего значения, взаимодействуя с его узлами через определенные адреса, при этом на время обращения она отключает прерывания, тем самым предотвращает переключение контекста между задачами. Две другие функции используют значение, полученное от chVTGetSystemTime, чтобы вычислить интервал между ним и входным параметром и чтобы сравнить его с заданными границами соответственно.

Таймер генерирует прерывания по заданному интервалу времени: когда таймер достигает установленного значения, он вызывает прерывание, которое приводит к выполнению обработчика прерывания. При возникновении прерывания управление передается в специализированный обработчик, в котором происходит обновление системного времени и обработка тайм-аутов.

2.3. Zephyr

В операционной системе реального времени Zephyr [4] так же реализован интерфейс взаимодействия с аппаратными таймерами. В

отличие от API таких таймеров в ChibiOS, Zephyr предоставляет больше способов для взаимодействия с ними: counter_start() для запуска таймера; counter_stop() для остановки работающего таймера; counter_get_value() для возврата текущего значения счетчика таймера; counter_set_channel_alarm() для установки будильника (alarm) на определенном канале таймера, она также позволяет задать событие, которое будет генерироваться по истечении заданного времени; counter_cancel_channel_alarm() для отмены ранее установленного будильника на указанном канале.

Инициализация и обработка прерываний таймеров в Zephyr осуществляется по схожему с ChibiOS принципу: в процессе инициализации системы происходит настройка таймеров, при этом определяются соответствующие прерывания, которые связываются с функцией для сброса соответствующих прерываний таймеров. Как и в ChibiOS, во время работы с узлами таймера прерывания отключаются, что предотвращает переключение контекста.

2.4. Вывод

В первую очередь таймеры необходимы для точного и надежного измерения временных интервалов. Все рассмотренные API операционных систем предоставляют эту возможность, однако интерфейсы в FreeRTOS и Zephyr имеют схожую структуру, в то время как в ChibiOS он отличается. Основываясь на проведенном обзоре, можно сделать вывод, что API должен предоставлять следующую функциональность для таймеров: инициализация, запуск и остановка, получение значения со счетчика и изменение режима работы.

3. Реализация

В данном разделе будет представлено реализованное API и структуры блока таймеров и самих таймеров.

3.1. API таймеров

На основе выводов, представленных в секции 2.4 было разработано следующее API:

- setup_timer() — функция для первоначальной настройки таймеров, которая в дальнейшем может быть использована для конфигурации параметров;
- get_timer(timer_index: u8) → Option<Timer> — получение экземпляра таймера по указанному индексу. Функция возвращает экземпляр таймера в случае успеха или None, если таймер занят или переданный индекс некорректен;
- loop_timer() — запуск отсчета программного таймера;
- start_timer() — активация аппаратного таймера;
- set_reload_mode(auto_reload: bool) — изменение режима работы таймера на циклический или однократный;
- change_period_timer(period: Duration) — изменение периода таймера;
- stop_condition_timer() → bool — остановка тиканья таймера, возвращает true в случае успеха и false, если устройство не поддерживает остановку счетчика;
- get_time() → Duration — получение текущего значение счетчика;
- release_timer() — освобождение аппаратного таймера.

Для работы с временными интервалами в системе был выбран тип Duration из стандартной библиотеки Rust[3]. Также были реализованы функции duration_to_ticks() и ticks_to_duration(), которые обеспечивают конвертацию между значениями типа Duration и типом TickType, используемым для представления времени в тиках в системе.

3.2. Структура блока таймеров и реализация функций из API

Блок таймеров содержит пять одинаковых таймеров, каждый из которых программируется независимо, а также конфигурационные регистры, служащие для разрешения или запрета на счёт каждого. Исходя из этого, структура каждого из таймеров одинакова и состоит из следующих полей:

- address — адрес регистров соответствующего таймера;
- duration — заданное значение времени для счетчика в тиках;
- resolution_mask — маска разрешения счета для соответствующего таймера, с помощью которой, используя побитовые операции, изменяется бит в управляющем регистре;
- in_use — индикатор, показывающий, используется ли таймер в данный момент;
- reload_mode — индикатор, указывающий, находится ли таймер в циклическом режиме или в режиме однократного срабатывания;
- lock_for_load — индикатор, показывающий, возможно ли начать загрузку нового значения в таймер;
- lock_for_now — индикатор, позволяющий определить, можно ли начать новый прием текущих тактов счетчика таймера;
- accessibility — методы чтения и записи байта по заданному адресу.

[3]https://doc.rust-lang.org/std/time/struct.Duration.html

Функции реализованы в соответствии с документацией, предоставленной консультантом.

3.3. Тестирование

Из-за ограниченного доступа к плате протестировать на ней написанный код не представилось возможным. Кроме того, аппаратные таймеры, для которых была разработана данная реализация, имеют специфическую архитектуру для процессора Комдив, что делает невозможным тестирование написанного кода с эмуляцией их поведения с помощью общедоступных эмуляторов. В связи с этим было принято решение использовать mock-тестирование, заменив реальный доступ к памяти при чтении и записи в регистры на фиктивный, что позволяет имитировать эти операции. Для этого был создан абстрактный интерфейс ByteAccess, который включает две функции: read_byte() и write_byte(). Реализация этого интерфейса передается в структуру таймера и хранится в поле accessibility. Таким образом, при вызове соответствующих методов из функций в реализации таймеров будет выполняться необходимая функциональность: при работе с реальными таймерами передается структура MemoryAccess, предоставляющая функции для прямого доступа к памяти, а при тестировании — структуры с необходимыми проверками и требуемыми возвращаемыми значениями.

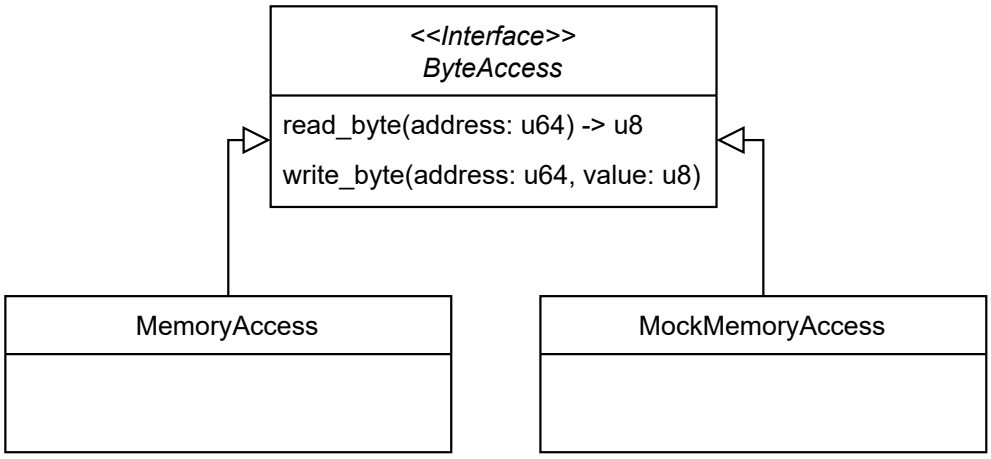

Рис. 1: Архитектура для mock-тестирования

4. Заключение

В ходе выполнения данной работы были достигнуты все поставленные задачи:

- Проведен обзор API таймеров трех существующих операционных систем реального времени;
- Улучшено существующее в системе API таймеров;
- Спроектирована структура блока таймеров;
- Реализованы функции из API.

Ссылка на pull request с реализацией[4].

[4]https://github.com/IvanArkhipov1999/Martos/pull/73

Список литературы

[1] ChibiOS. Real-time operating system for embedded systems. — Access mode: https://www.chibios.org/dokuwiki/doku.php (online; accessed: 29 декабря 2024 г.).

[2] Embedded Survey 2023: More IP Reuse as Workloads Surge. — Access mode: https://www.embedded.com/embedded-survey-2023-more-ip-reuse-as-workloads-surge/ (online; accessed: 29 декабря 2024 г.).

[3] FreeRTOS. Real-time operating system for microcontrollers and small microprocessors. — Access mode: https://freertos.org/ (online; accessed: 29 декабря 2024 г.).

[4] Zephyr. A proven RTOS ecosystem. — Access mode: https://www.zephyrproject.org/ (online; accessed: 29 декабря 2024 г.).