Операционная система реального времени FX-RTOS
Руководство по сборке и установке

Версия 3.X

Оглавление

Описание

FX-RTOS – это ядро ОСРВ, предназначенное для устройств на микроконтроллерах и микропроцессорах. Ядро написано на языке Си стандарта C99.

Функции

API включает в себя следующие сервисы:

- Потоки;
- программные таймеры (одноразовые и периодические);
- семафоры;
- мьютексы защитой от инверсии приоритета;
- очереди сообщений;
- пулы блоков памяти;
- события;
- условные переменные;
- барьеры;
- пулы памяти произвольного размера;
- rw-блокировки.

Поддерживаемое оборудование и инструментальные средства

FX-RTOS поддерживает наиболее распространенные процессорные архитектуры:

- ARM Cortex-M3 + (архитектура ARMv7-M, также может использоваться на ARMv8-M);
- RISC-V (профиль RV32I);
- МЦСТ Эльбрус 2000;
- Intel/AMD X86_64;
- MIPS32 M4K;
- TI MSP430;
- AVR32.

Типовые конфигурации:

Релиз	Платформа
`async-cortex-m3`	Минимальная ОС, использующая процедуры обработки событий вместо потоков, для cortex-m3+
`standard-cortex-m3`	Обычная конфигурация для контроллеров cortex-m3+
`standard-riscv32i`	Обычная конфигурация для набора команд RISC-V rv32i

Версия ARM поддерживает компиляторы GCC, Keil MDK и IAR EWARM. Версия RISC-V на данный момент поддерживает только GCC.

В качестве ОС для сборки может использоваться Windows или Linux (Mac тоже должен работать, но не тестировался). Никаких внешних зависимостей, кроме компилятора, не требуется.

С чего начать

Этот репозиторий содержит несконфигурированную версию ядра, представленную в виде набора компонентов - исходных текстов. Это полезно, если вы хотите создать специфическую конфигурацию или внести свой вклад в развитие ОС. В случае, если вам просто нужно готовое ядро для использования в своём приложении, подумайте об использовании готовых сборок предварительно сконфигурированных ядер, доступных для ARM и RISC-V.

Как собрать библиотеку из предварительно настроенных источников:

- Загрузите и распакуйте соответствующий архив выпуска из каталога готовых сборок;
- Убедитесь, что поддерживаемый компилятор доступен через переменную окружения PATH;
- Установите GCC_PREFIX в качестве префикса компилятора, если вы используете GCC (например 'riscv-none-embed-');
- Войдите в каталог, в котором находится build.bat;
- Запускаем build.bat.

Для тех, кто не хочет возиться с инструментами и исходным кодом, мы предоставляем готовые двоичные файлы. Хотя бинарная версия может быть достаточной для большинства пользователей, в ней отсутствуют параметры настройки и оптимизации.

Как собрать библиотеку с нуля:

- Убедитесь, что скрипт fx-dj.py из каталога fx-dj доступен через переменную окружения PATH.
- Убедитесь, что поддерживаемый компилятор доступен через PATH.
- Установить переменные окружения:
    - FXRTOS_DIR как путь к корневой папке ядра.
    - GCC_PREFIX в качестве префикса компилятора, если вы используете GCC (например, 'arm-none-eabi-' для ARM).
    - FXDJ как инструмент внедрения зависимостей (например, fx-dj.py).
- Введите каталог для целевого ядра (например, cores/standard-cortex-m3).
- Запустите build.bat в Windows или make src, а затем make lib в Linux / Mac (только ARM).

Как использовать

ОС может быть связана с проектом как двоичный файл или может быть добавлена как набор исходных файлов в проект IDE. Демонстрации использования находятся в каталоге примеров. Процесс установки ОСРВ на каждый тип устройство индивидуален, за инструкциями обращайтесь в руководство на аппаратное обеспечение.

Ограничения

При разработке приложений с критичным временем задержки следует учитывать следующие ограничения:

- Широковещательная передача (или "уведомление всех") объектов синхронизации, таких как condvar или event. Большее количество ожидающих потоков приводит к увеличению задержки, поскольку процесс уведомления не является вытесняющим. Возможные решения: не использовать широковещательные объекты или ограничивать максимальное количество ожидающих потоков.
- Политика уведомления на основе приоритетов с объектами синхронизации (т. Е. Отправка семафора и освобождение ожидающего потока с наиболее приоритетным приоритетом) использует линейный поиск с отключенным планированием. Это означает, что N ожидающих потоков приводит к неограниченному планированию O (n) и задержке прерывания. Возможные решения: используйте политику уведомления FIFO или ограничьте

максимальное количество ожидающих потоков для любого объекта синхронизации.

- Очистка очереди сообщений освобождает до N потоков, где N - длина очереди. Возможные решения: не используйте очистку очереди или ограничьте длину очереди разумным значением.
- Реализация таймеров LW использует сортированные очереди и линейный поиск, что дает задержку O(n) в зависимости от количества активных таймеров в системе. Возможные решения: Ограничьте максимальное количество программных таймеров и не используйте временные интервалы.