Операционная система Eremex FX-RTOS
Описание прикладного интерфейса
Версия 3.0

Содержание

FX-RTOS_API_ref_ru

Операционная система FX-RTOS
Описание прикладного интерфейса
Версия 3.0

Правовая информация

Внимание!

В случае возникновения вопросов по использованию программ FX-RTOS, Delta Design, TopoR, SimOne, пожалуйста, обращайтесь:

Форум «ЭРЕМЕКС»: http://forum.eremex.ru/

Техническая поддержка
E-mail: support@eremex.ru
Skype: supporteremex

Отдел продаж
Тел. +7 (495) 232-18-64
E-mail: info@eremex.ru
E-mail: sales@eremex.ru

Введение

Об этом руководстве

Настоящий документ описывает интерфейс прикладного программирования (API) для OCPB FX-RTOS. Для подробного описания функциональных компонентов OCPB и принципов их работы обратитесь к руководству пользователя FX-RTOS.

Терминология

DPC – Deferred procedure call. Процедура, выполняемая как отложенная часть обработки прерывания (при ее обработке аппаратные прерывания разрешены).

HAL – Hardware abstraction layer. Набор функций, реализация которых может меняться, при портировании OCPB на другую аппаратную платформу.

Асинхронное событие - изменение счетчика инструкций, произошедшее не в результате действий приложения (например, аппаратное прерывание).

SPL – System priority level. Уровень приоритета кода, выполняющегося в данный момент, по отношению к асинхронным событиям.

Многопроцессорная система – система, содержащая более одного логического процессора (под это определение подпадают как многопроцессорные, так и многоядерные системы).

Поток – независимая последовательность инструкций, имеющая приоритет, счетчик инструкций и стек.

Миграция потока – процесс, по завершению которого, поток, выполнявшийся на одном из процессоров, начинает выполнение на другом процессоре.

Программное прерывание – прерывание, которое было инициировано программно операционной системой и не связано с работой аппаратного обеспечения.

Фрейм прерывания – структура, формируемая на стеке выполнявшегося потока, в результате прерывания. Содержит состояние регистров, что позволяет его восстановить, после завершения обработки прерывания.

Контекст исполнения – характеристика окружения, которая накладывает ограничения (времени исполнения) на исполняемый код.

Блокировка потока (или ожидание) – приостановка исполнения потока до выполнения определенного условия.

Формат описания функций API

Описание функции API включает в себя следующие основные разделы:

- **Прототип**
  Данный раздел содержит прототип описываемой функции на языке C в том виде, в котором она описана в заголовочных файлах ОС.
- **Описание**
  После прототипа приводится краткое описание функционала данной функции.
- **Аргументы**
  Содержит список аргументов функции (в том виде, как он описан в разделе «прототип») и их краткие описания и/или возможные значения.
- **Возвращаемое значение**
  Для функций, возвращающих значение, оно описывается в данном разделе. Если возвращаемое значение имеет определенные символьные имена (например, коды ошибок и их символьные значения) – приводится описание для каждого из них.
- **Контекст вызова**
  OCPB имеет несколько контекстов выполнения (обработчики прерываний, потоки и т.д.), каждая функция API предназначена для выполнения в определенном контексте. Например, архитектура OCPB запрещает блокирующие (вызывающие блокировку потока) вызовы в любом контексте кроме пользовательских потоков, это значит, что любые функции, которые прямо или косвенно вызывают блокировку нельзя использовать внутри обработчиков прерываний. Контекст вызова характеризуется главным образом с помощью уровня SPL, на котором происходит вызов функции. Для подробного описания уровней SPL и их значения обратитесь к руководству пользователя FX-RTOS.
- **Ремарки**
  Данный раздел содержит описание особых сценариев использования функции, а также дополнительную информацию для разработчиков.
- **Пример**
  Пример использования функции.

Интерфейсы HAL

В данном разделе описываются функции уровня абстракции оборудования (HAL), которые являются наиболее низкоуровневым кроссплатформенным интерфейсом.

Поддержка многопроцессорных систем

hal_mp_get_cpu_count

```
unsigned int hal_mp_get_cpu_count(void);
```

Описание

Получение актуального количества процессоров в системе.

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Нет.

Возвращаемое значение

Количество процессоров в системе.

Ремарки

Нет.

hal_mp_get_current_cpu

```
unsigned int hal_mp_get_current_cpu(void);
```

Описание

Получение номера текущего процессора.

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Нет.

Возвращаемое значение

Номер текущего процессора.

Ремарки

В многопроцессорной системе миграции потоков могут возникать в любой момент на уровне SPL_LOW, поэтому если функция вызвана на уровне SPL_LOW, ее результат может стать неактуальным во время выполнения этой функции. Для гарантированной актуальности полученного значения следует использовать функцию на уровне выше SPL_LOW.

hal_mp_request_ipi

```
void hal_mp_request_ipi(unsigned int pu, spl_t level);
```

Описание

Запрос межпроцессорного прерывания указанного уровня.

Контекст вызова

SPL > LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
cpu	Процессор (от 0 до значения, возвращаемого hal_mp_get_cpu_count) для которого следует выполнить запрос соответствующего прерывания.
level	Уровень программного прерывания, которое следует запросить. В текущей реализации ядра поддерживаются только прерывания уровня SPL_DISPATCH, поэтому значение игнорируется.

Возвращаемое значение

Нет.

Ремарки

В однопроцессорных системах, данная функция может быть отображена на hal_async_request_swi, поскольку любой запрос прерывания эквивалентен запросу прерывания для текущего (и единственного) процессора. Поэтому, хотя некоторые реализации могут и не иметь этого ограничения, рекомендуется распространять на эту функцию ограничения, свойственные hal_async_request_swi – вызывать ее только в блоке, где прерывания соответствующего уровня замаскированы.

Управление SPL

hal_async_get_current_spl

```
spl_t hal_async_get_current_spl(void);
```

Описание

Получение текущего уровня SPL.

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Нет.

Возвращаемое значение

Текущий уровень SPL.

Ремарки

Нет.

hal_async_lower_spl

```
void hal_async_lower_spl(spl_t new_spl);
```

Описание

Понижение уровня SPL.

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
new_spl	Значение, которое было возвращено предыдущим вызовом hal_async_raise_spl.

Возвращаемое значение

Нет.

Ремарки

В связи с различными возможными реализациями HAL, требуется обеспечивать парность вызовов hal_async_raise_spl/hal_async_lower_spl. Если в блоке с повышенным SPL были запрошены программные или аппаратные прерывания, исполнение будет прервано после выполнения данной функции.

hal_async_raise_spl

```
spl_t hal_async_raise_spl(spl_t new_spl);
```

Описание

Установка текущего уровня SPL процессора. Устанавливаемый SPL не может быть меньше (менее приоритетный) чем текущий.

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
new_spl	Новый SPL, который необходимо установить. Множество возможных значений определяется HAL, обязательно должны поддерживаться 3 уровня: 1. SPL_SYNC – запрет всех асинхронных активностей на данном процессоре (аппаратных и программных прерываний). 2. SPL_DISPATCH – запрет программных прерываний и диспетчеризации (аппаратные прерывания остаются разрешенными). 3. SPL_LOW – все асинхронные активности разрешены. Уровень приложений.

Возвращаемое значение

Предыдущее значение SPL, которое необходимо передавать в функцию hal_async_lower_spl.

Ремарки

Возвращаемое значение является некоторым контекстом, который должен быть восстановлен, при выходе из блока с повышенным SPL. В зависимости от платформы, оно может не совпадать с фактическим SPL, который был в момент вызова функции. Вызывающий код не должен анализировать это значение, допускается только его передача в функцию hal_async_lower_spl. Для получения актуального SPL следует использовать функцию hal_async_get_current_spl.
В зависимости от конфигурации, если используется унифицированная схема синхронизации прерываний, уровни SPL_DISPATCH и SPL_SYNC могут численно совпадать, в этом случае существует только два уровня – SPL_LOW – на котором работают приложения, и SPL_DISPATCH /SPL_SYNC на которых прерывания и другие асинхронные активности запрещены.
В связи с различными возможными реализациями HAL требуется обеспечивать парность вызовов hal_async_raise_spl/hal_async_lower_spl.

hal_async_request_swi

```
void hal_async_request_swi(spl_t level);
```

Описание

Запрос программного прерывания соответствующего уровня для данного процессора.

Контекст вызова

SPL > LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
level	Уровень программного прерывания, которое следует запросить. В текущей реализации ядра поддерживаются только прерывания уровня SPL_DISPATCH, поэтому значение игнорируется.

Возвращаемое значение

Нет.

Ремарки

Не рекомендуется запрашивать программные прерывания, когда они не замаскированы из-за возможных особенностей их реализации (программной или аппаратной). Запрос прерывания всегда должен осуществляться в блоке, где прерывания этого уровня замаскированы. Поведение системы при запросе программных прерываний в блоке, где они размаскированы, не определено и зависит от реализации.

Управление прерываниями

hal_intr_get_current_vect

```
unsigned int hal_intr_get_current_vect(void);
```

Описание

Получение текущего вектора прерывания.

Контекст вызова

SPL > DISPATCH

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✗
Обработчик прерывания	✓
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Нет.

Возвращаемое значение

Текущий вектор прерывания.

Ремарки

Функция должна вызываться только в контексте обработчиков аппаратных прерываний, при вызове в другом контексте (в т.ч. в обработчике программных прерываний) результат ее выполнения не определен.

Фрейм прерывания и контекст

hal_intr_frame_alloc

```
hal_intr_frame_t* hal_intr_frame_alloc(hal_intr_frame_t* base);
```

Описание

Выделение нового фрейма прерывания на базе указанного.

Контекст вызова

SPL = DISPATCH

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✗
Обработчик прерывания	✗
Отложенная процедура/DPC	✓
Обработчик таймера	✗

Аргументы

Базовый фрейм, относительно которого будет выделен новый блок.

Возвращаемое значение

Указатель на выделенный и инициализированный фрейм прерывания.

Ремарки

Функция должна вызываться только в контексте обработчиков программных прерываний, при вызове в другом контексте результат ее выполнения не определен.

hal_intr_frame_get

```
hal_intr_frame_t* hal_intr_frame_get(void);
```

Описание

Получение текущего указателя фрейма.

Контекст вызова

SPL = DISPATCH

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✗
Обработчик прерывания	✗
Отложенная процедура/DPC	✓
Обработчик таймера	✗

Аргументы

Нет.

Возвращаемое значение

Указатель на текущий фрейм.

Ремарки

Функция должна вызываться только в контексте обработчиков программных прерываний, при вызове в другом контексте результат ее выполнения не определен.

hal_intr_frame_modify

```
void hal_intr_frame_modify(hal_intr_frame_t* frame, int reg, uintptr_t val);
```

Описание

Модификация фрейма прерывания.

Контекст вызова

SPL = DISPATCH

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✗
Обработчик прерывания	✗
Отложенная процедура/DPC	✓
Обработчик таймера	✗

Аргументы

Аргумент	Описание
frame	Фрейм для модификации.
reg	Регистр, который необходимо модифицировать. Допустимые значения определяются конкретным HAL, среди обязательных должны поддерживаться только следующие: KER_FRAME_ENTRY – модификация привилегированного счетчика инструкций (для kernel-mode). KER_FRAME_ARG0 – аргумент функции, вызываемой как KER_FRAME_ENTRY.
val	Значение указанного регистра.

Возвращаемое значение

Нет.

Ремарки

Функция должна вызываться только в контексте обработчиков программных прерываний, при вызове в другом контексте результат ее выполнения не определен.

hal_intr_frame_set

```
void hal_intr_frame_set(hal_intr_frame_t* frame);
```

Описание

Установка текущего фрейма прерывания.

Контекст вызова

SPL >= DISPATCH

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✗
Обработчик прерывания	✗
Отложенная процедура/DPC	✓
Обработчик таймера	✗

Аргументы

Аргумент	Описание
frame	Фрейм для установки.

Возвращаемое значение

Нет.

Ремарки

Функция должна вызываться только в контексте обработчиков программных прерываний, при вызове в другом контексте результат ее выполнения не определен.

Функции прикладного интерфейса приложений FX-RTOS

Block pool

Объект используется для выделения блоков памяти фиксированного размера. Пользователь предоставляет выделенный заранее буфер, который используется как источник памяти.

fx_block_pool_alloc

```
int fx_block_pool_alloc(fx_block_pool_t* bp, void** blk, fx_event_t* ev);
```

Описание

Выделяет блок памяти фиксированного размера из пула. Если пул не содержит свободных блоков – выполнение вызывающего потока приостанавливается до тех пор, пока они не появятся в результате освобождения другими потоками. Для отмены ожидания может использоваться отменяющее событие.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	❌
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
bp	Указатель на объект «пул блоков памяти», из которого требуется выделить блок.
blk	Адрес переменной, где будет размещен адрес выделенного блока в случае успешного завершения функции.
ev	Опциональный указатель на отменяющее событие.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BLOCK_POOL_OK	Успешное завершение.
FX_BLOCK_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_BLOCK_POOL_INVALID_OBJ	Некорректно проинициализированный объект.
FX_THREAD_WAIT_DELETED	Пул блоков (или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_THREAD_WAIT_CANCELLED	Ожидание было прервано из-за установки отменяющего события в активное состояние.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

Нет.

Пример

```
fx_block_pool_t bp;
void my_func(void)
{
    void* allocated_block = NULL;
    int status = fx_block_pool_alloc(&bp, &allocated_block, NULL);
}
```

fx_block_pool_avail_blocks

```
int fx_block_pool_avail_blocks(fx_block_pool_t* bp, uint32_t* count);
```

Описание

Получение количества свободных блоков, содержащихся в пуле.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
bp	Объект «пул блоков памяти»,количество свободныз блоков в котором требуется получить.
count	Указатель на переменную, в которой будет сохранено количество свободных блоков в пуле.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BLOCK_POOL_OK	Успешное завершение.
FX_BLOCK_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект или на переменную-приемник.
FX_BLOCK_POOL_INVALID_OBJ	Некорректный блок (не выделенный ранее с помощью функций выделения блоков).

Ремарки

Нет.

Пример

```
fx_block_pool_t bp;
void my_func(void)
{
    uint32_t blocks_count = 0;
    int status = fx_block_pool_avail_blocks(&bp, &blocks_count);
}
```

fx_block_pool_deinit

```
int fx_block_pool_deinit(fx_block_pool_t* bp);
```

Описание

Удаляет объект «пул блоков памяти».

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
bp	Указатель на объект «пул блоков памяти», который должен быть удален.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BLOCK_POOL_OK	Успешное завершение.
FX_BLOCK_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_BLOCK_POOL_INVALID_OBJ	Некорректно проинициализированный объект.

Ремарки

Нет.

Пример

```
fx_block_pool_t bp;
void my_func(void)
{
    int status = fx_block_pool_deinit(&bp);
}
```

fx_block_pool_init

```
int fx_block_pool_init(fx_block_pool_t* bp, void* base, size_t sz, size_t blk_sz, const fx_sync_policy_t p);
```

Описание

Инициализирует объект «пул блоков памяти».

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
bp	Указатель на объект «пул блоков памяти», который надо проинициализировать.
base	Базовый адрес непрерывного блока памяти (выделенного статически, либо с помощью динамического аллокатора) в котором будут размещаться блоки, входящие в пул. Данный указатель должен быть выровнен как минимум на размер типа uintptr_t.
sz	Размер памяти по адресу base.
blk_sz	Размер одного блока. Размер блока округляется до размера типа unitptr_t.
p	Политика уведомления ожидающих потоков. Определяется используемым слоем синхронизации. По умолчанию должно использоваться значение FX_SYNC_POLICY_DEFAULT. В случае явного указания политики могут поддерживаться следующие значения: 1. FX_SYNC_POLICY_FIFO – уведомление ожидающих потоков в том порядке, в котором они начали ожидание. 2. FX_SYNC_POLICY_PRIO – уведомление в первую очередь ожидателей с максимальным приоритетом. Так как поддержка различных политик зависит от конфигурации ОС, рекомендуется использовать значение по-умолчанию в тех случаях, когда требуется достичь максимальной переносимости между различными конфигурациями.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BLOCK_POOL_OK	Успешное завершение.
FX_BLOCK_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_BLOCK_POOL_NO_MEM	Недопустимый указатель base или размер памяти недостаточен для размещения хотя бы одного блока указанного размера (блок содержит заголовок, поэтому его фактический размер может быть больше, чем blk_sz).

FX_BLOCK_POOL_UNSUPPORTED_POLICY	Неподдерживаемая политика нотификации.
FX_BLOCK_POOL_IMPROPER_ALIGN	Указатель base имеет неверное выравнивание.

Ремарки

Нет.

Пример

```
fx_block_pool_t bp;
int arr[200];
void my_func(void)
{
    int status = fx_block_pool_init(&bp, arr, sizeof(arr), 20, FX_SYNC_POLICY_FIFO);
}
```

fx_block_pool_release

```
int fx_block_pool_release(void* blk_ptr);
```

Описание

Возвращает блок памяти (выделенный ранее) в пул.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
blk_ptr	Указатель на блок памяти, который требуется вернуть в пул. Он должен быть выделен ранее из пула с помощью функций fx_block_pool_alloc или fx_block_pool_timedalloc.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BLOCK_POOL_OK	Успешное завершение.
FX_BLOCK_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_BLOCK_POOL_INVALID_OBJ	Некорректный блок (не выделенный ранее с помощью функций выделения блоков).

Ремарки

Нет.

Пример

```
fx_block_pool_t bp;
void my_func(void)
{
    void* allocated_block = NULL;
    int status = fx_block_pool_timedalloc(&bp, &allocated_block, 20);
    if(status == FX_BLOCK_POOL_OK)
    {
        fx_block_pool_release(allocated_block);
    }
}
```

fx_block_pool_timedalloc

```
int fx_block_pool_timedalloc(fx_block_pool_t* bp, void** blk, uint32_t tout);
```

Описание

Выделяет блок памяти фиксированного размера из пула. Если пул не содержит свободных блоков – выполнение вызывающий поток приостанавливается до тех пор, пока они не появятся в результате особождения другими потоками. Ожидание может быть прерывано досрочно из-за истечения таймаута.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
bp	Указатель на объект «пул блоков памяти», из которого требуется выделить блок.
blk	Адрес переменной, где будет размещен адрес выделенного блока в случае успешного завершения функции.
tout	Таймаут ожидания в тиках таймера. Для указания бесконечного таймаута следует использовать значение FX_THREAD_INFINITE_TIMEOUT. Также значение таймаута не должно превышать величину FX_TIMER_MAX_RELATIVE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BLOCK_POOL_OK	Успешное завершение.
FX_BLOCK_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_BLOCK_POOL_INVALID_OBJ	Некорректно проинициализированный объект.
FX_THREAD_WAIT_DELETED	Пул блоков (или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_THREAD_WAIT_TIMEOUT	Ожидание было прерывано из-за истечения таймаута.
FX_THREAD_INVALID_TIMEOUT	Значение таймаута не равно FX_THREAD_INFINITE_TIMEOUT и превышает величину FX_TIMER_MAX_RELATIVE_TIMEOUT.

Ремарки

Нет.

Пример

```
fx_block_pool_t bp;
void my_func(void)
{
    void* allocated_block = NULL;
    int status = fx_block_pool_timedalloc(&bp, &allocated_block, 20);
}
```

DPC

Отложенные вызовы процедур (deferred procedure calls) являются основным механизмом взаимодействия между обработчиками прерываний и потоками в сегментированной архитектуре прерываний ядра.

fx_dpc_cancel

```
bool fx_dpc_cancel(fx_dpc_t* dpc);
```

Описание

Извлекает из очереди DPC-объект, если он находился в очереди.

Контекст вызова

SPL >= DISPATCH

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
dpc	Указатель на DPC-объект.

Возвращаемое значение

true, если перед вызовом этой функции объект действительно находился в очереди DPC, false в противоположном случае.

Ремарки

После выполнения этой функции не гарантируется, что отложенная процедура завершила выполнение.

Пример

```
fx_dpc_t dpc;
void* my_thread(void arg)
{
    fx_dpc_cancel(&dpc);
    //...
}
```

fx_dpc_init

```
void fx_dpc_init(fx_dpc_t* dpc);
```

Описание

Инициализация объекта DPC. В качестве процессора, ассоциированного с DPC устанавливается текущий (на котором вызвана данная функция).

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
dpc	Указатель на DPC-объект для инициализации.

Возвращаемое значение

Нет.

Ремарки

Нет.

Пример

```
fx_dpc_t dpc;
void my_thread(void* arg)
{
    fx_dpc_init(&dpc);
    //...;
}
```

fx_dpc_request

```
bool fx_dpc_request(fx_dpc_t* dpc, void (*func)(fx_dpc_t*, void*), void* arg);
```

Описание

Вставляет DPC объект в очереди DPC для процессора, который был ассоциирован с DPC-объектом.

Контекст вызова

SPL >= DISPATCH

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
dpc	Указатель на DPC-объект.
func	Отложенная процедура, которая будет вызвана как часть обработки прерывания диспетчеризации при понижении уровня SPL ниже SPL_DISPATCH.
arg	Аргумент, который будет передан в отложенную процедуру.

Возвращаемое значение

true – если объект был добавлен в очереди, false – в противном случае.

Ремарки

Если вставка объекта была произведена для текущего процессора, после успешного завершения этой функции будет запрошено программное прерывание уровня DISPATCH. Если вставка была произведена в очереди другого процессора – будет запрошено межпроцессорное прерывание (IPI) того же уровня.
Если функция вызвана для объекта, который уже добавлен в какую-либо очередь, никаких действий производиться не будет.
Реализация DPC не гарантирует SPL=DISPATCH в момент ее исполнения. Если SCHED_LEVEL=SYNC (унифицированная архитектура прерываний), DPC-процедура может быть вызвана в контексте обработчика прерывания, так как в этом случае возможно непосредственное использование примитивов синхронизации из обработчиков и очередь отложенных вызовов не требуется.

Пример

```
fx_dpc_t dpc;

void my_dpc_func(fx_dpc_t* dpc, void* arg)
{
    // Arg is 0x11223344 here.
}

void my_thread(void* arg)
{
    spl_t prev = hal_async_raise_spl(SPL_DISPATCH);
    //
    // By default, init sets CPU associated with the DPC as ours.
    //
    fx_dpc_init(&dpc);
    //
    // Insert DPC object into the queue.
    //
    fx_dpc_request(&dpc, my_dpc_func, 0x11223344);
    //
    // DISPATCH interrupt will be handled at end of this function,
    // when the SPL lowers.
    //
    hal_async_lower_spl(prev);
    //...;
}
```

fx_dpc_set_target_cpu

```
void fx_dpc_set_target_cpu(fx_dpc_t* dpc, unsigned int cpu);
```

Описание

Устанавливает процессор, ассоциированный с DPC-объектом.

Контекст вызова

SPL >= DISPATCH

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
dpc	Указатель на DPC-объект.
cpu	Целевой процессор, с которым должен быть ассоциирован объект. Допустимые значения от 0 до HAL_MP_CPU_MAX-1.

Возвращаемое значение

Нет.

Ремарки

В последующих вызовах fx_dpc_request вставка всегда производится в очереди того процессора, с которым ассоциирован объект, даже если фактически вставка производится с другого процессора.
Объект не должен использоваться во время вызова этой функции.

Пример

```
fx_dpc_t dpc;
void my_thread(void* arg)
{
    //
    // Set CPU 1 as CPU associated with the given DPC object.
    //
    fx_dpc_set_target_cpu(&dpc, 1);
    //...;
}
```

Memory pool

Пул памяти используется для выделения блоков памяти произвольного размера. В зависимости от реализации данный компонент может вызывать задержки связанные с фрагментацией, поэтому в приложениях жёсткого реального времени рекомендуется использовать блоки памяти фиксированного размера.

fx_mem_pool_add_mem

```
int fx_mem_pool_add_mem(fx_mem_pool_t* pool, uintptr_t base, size_t size);
```

Описание

Добавление памяти в пул. Добавленная в пул память может использоваться только после корректного ее выделения средствами пула. Данная функция может использоваться многократно, если необходимо добавить в пул несколько несмежных блоков памяти.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
pool	Указатель на объект «пул памяти», в который добавляется память.
base	Базовый адрес блока памяти, который следует добавить в пул.
size	Размер блока памяти, добавляемого в пул.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MEM_POOL_OK	Успешное завершение.
FX_MEM_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_MEM_POOL_INVALID_BUF	Не выровненный адрес блока памяти.

Ремарки

Нет.

Пример

```
fx_mem_pool_t pool;
int array[200];

void my_func(void)
{
    int status = fx_mem_pool_add_mem(&pool, array, sizeof(array));
}
```

fx_mem_pool_alloc

```
int fx_mem_pool_alloc(fx_mem_pool_t* pool, const size_t sz, void** ptr);
```

Описание

Выделение памяти из пула.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
pool	Указатель на объект «пул памяти», из которого выделяется память.
sz	Размер выделяемого блока.
ptr	Указатель на переменную, в которой будет размещен адрес выделенного блока (при успешном завершении функции). При неуспешном завершении значение по данному адресу не изменяется.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MEM_POOL_OK	Успешное завершение.
FX_MEM_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект или указатель на переменную-приемник.
FX_MEM_POOL_INVALID_OBJ	Некорректно проинициализированный объект.
FX_MEM_POOL_ZERO_SZ	Нулевой размер выделяемой памяти.
FX_MEM_POOL_NO_MEM	В пуле недостаточно памяти для выделения блока заданного размера.

Ремарки

Нет.

Пример

```
fx_mem_pool_t pool;

void my_func(void)
{
    void* allocated_chunk;
    //
    // Try allocate 20 bytes from pool. If succeeded, the pointer holds
    // address of allocated block.
    //
    int status = fx_mem_pool_alloc(&pool, 20, &allocated_chunk);
}
```

fx_mem_pool_deinit

```
int fx_mem_pool_deinit(fx_mem_pool_t* pool);
```

Описание

Удаление объекта «пул памяти». Память, которая содержалась в пуле, может быть использована для любых других целей. Никакие функции работы с пулом не должны вызываться над объектом после его удаления.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
pool	Указатель на объект «пул памяти», который надо удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MEM_POOL_OK	Успешное завершение.
FX_MEM_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_MEM_POOL_INVALID_OBJ	Некорректно проинициализированный объект.

Ремарки

Нет.

Пример

```
fx_mem_pool_t pool;

void my_func(void)
{
    int status = fx_mem_pool_deinit(&pool);
}
```

fx_mem_pool_free

```
int fx_mem_pool_free(fx_mem_pool_t* pool, void* addr);
```

Описание

Возвращение в пул памяти, выделенной ранее с помощью fx_mem_pool_alloc .

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
pool	Указатель на объект «пул памяти», в который возвращается выделенный ранее блок.
addr	Адрес блока, возвращаемого в пул.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MEM_POOL_OK	Успешное завершение.
FX_MEM_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект или возвращаемый блок.
FX_MEM_POOL_INVALID_OBJ	Некорректно проинициализированный объект или некорректный (не выделенный ранее из данного пула) блок.

Ремарки

Нет.

Пример

```
fx_mem_pool_t pool;

void my_func(void)
{
    void* allocated_chunk;
    //
    // Try allocate 20 bytes from pool. If succeeded, the pointer holds
    // address of allocated block.
    //
    int alloc_status = fx_mem_pool_alloc(&pool, 20, &allocated_chunk);
    int free_status = fx_mem_pool_free(&pool, allocated_chunk);
}
```

fx_mem_pool_get_max_free_chunk

```
int fx_mem_pool_get_max_free_chunk(fx_mem_pool_t* pool, size_t* max_chunk);
```

Описание

Получение размера максимального непрерывного блока памяти, из содержащихся в пуле.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
pool	Указатель на объект «пул памяти».
max_chunk	Указатель на переменную, в которой будет сохранен размер максимального непрерывного блока, из содержащихся в пуле.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MEM_POOL_OK	Успешное завершение.
FX_MEM_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект или указатель на переменную-приемник.
FX_MEM_POOL_INVALID_OBJ	Некорректно проинициализированный объект.

Ремарки

Нет.

Пример

```
fx_mem_pool_t pool;

void my_func(void)
{
    size_t max_chunk;
    int status = fx_mem_pool_get_max_free_chunk(&pool, &max_chunk);
}
```

fx_mem_pool_init

```
int fx_mem_pool_init(fx_mem_pool_t* pool);
```

Описание

Инициализирует объект «пул памяти». Пул всегда инициализируется пустым; память, которой он должен управлять следует добавить в него одним или несколькими вызовами функции fx_mem_pool_add_mem.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
pool	Указатель на объект «пул памяти», который надо проинициализировать.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MEM_POOL_OK	Успешное завершение.
FX_MEM_POOL_INVALID_PTR	Недопустимый (нулевой) указатель на объект.

Ремарки

Нет.

Пример

```
fx_mem_pool_t pool;

void my_func(void)
{
    int status = fx_mem_pool_init(&pool);
}
```

RW-блокировки (опциональный компонент)

fx_rwlock_deinit

```
int fx_rwlock_deinit(fx_rwlock_t* rw);
```

Описание

Удаление RW-блокировки. Все потоки, заблокированные на rw-блокировке возобновляют выполнение со статусом FX_THREAD_WAIT_DELETED.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	❌
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
rw	Указатель на RW-блокировку, которую надо удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_RWLOCK_OK	Успешное завершение.
FX_RWLOCK_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_RWLOCK_INVALID_OBJ	Объект не был ранее корректно проинициализирован функцией fx_rwlock_init.

Ремарки

В результате выполнения данной функции вызывающий поток может быть вытеснен.

Пример

```
fx_rwlock_t rwlock;

void my_thread(void* arg)
{
    //
    // Delete the rwlock.
    //
    fx_rwlock_deinit(&rwlock);
    //...;
}
```

fx_rwlock_init

```
int fx_rwlock_init(fx_rwlock_t* rw, const fx_sync_policy_t policy);
```

Описание

Инициализация RW-блокировки.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
rw	Указатель на RW-блокировку, которую надо проинициализировать.
policy	Политика уведомления ожидающих потоков. Определяется используемым слоем синхронизации. По умолчанию должно использоваться значение FX_SYNC_POLICY_DEFAULT. В случае явного указания политики могут поддерживаться следующие значения: 1. FX_SYNC_POLICY_FIFO – уведомление ожидающих потоков в том порядке, в котором они начали ожидание. 2. FX_SYNC_POLICY_PRIO – уведомление в первую очередь ожидателей с максимальным приоритетом. Так как поддержка различных политик зависит от конфигурации ОС, рекомендуется использовать значение по-умолчанию в тех случаях, когда требуется достичь максимальной переносимости между различными конфигурациями.

Возвращаемое значение

Возвращаемое значение	Описание
FX_RWLOCK_OK	Успешное завершение.
FX_RWLOCK_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_RWLOCK_UNSUPPORTED_POLICY	Некорректное значение параметра policy.

Ремарки

Нет.

Пример

```
fx_rwlock_t rwlock;

void my_thread(void* arg)
{
    //
    // Initialize the rwlock.
    //
    fx_rwlock_init(&rwlock, FX_SYNC_POLICY_DEFAULT);
    // ...;
}
```

fx_rwlock_rd_lock

```
int fx_rwlock_rd_lock(fx_rwlock_t* rw, fx_event_t* cancel_event);
```

Описание

Блокировка читателем с опциональным отменяющим событием. RW-блокировка допускает одновременный захват объекта несколькими читателями, если объект не заблокирован писателем.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
rw	Указатель на RW-блокировку, которую надо захватить для чтения.
cancel_event	Отменяющее событие. Может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_RWLOCK_OK	Успешное завершение.
FX_RWLOCK_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_RWLOCK_INVALID_OBJ	Объект не был ранее корректно проинициализирован функцией fx_rwlock_init.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.
FX_THREAD_WAIT_CANCELLED	Ожидание было прервано из-за установки отменяющего события в активное состояние.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

В результате выполнения данной функции вызывающий поток может быть вытеснен.

Пример

```
fx_rwlock_t rwlock;

void my_thread(void* arg)
{
    //
    // Acquire the rwlock as a reader.
    //
    fx_rwlock_rd_lock(&rwlock, NULL);
    // ...;
}
```

fx_rwlock_rd_timedlock

```
int fx_rwlock_rd_timedlock(fx_rwlock_t* rw, uint32_t timeout);
```

Описание

Блокировка читателем с опциональным таймаутом. RW-блокировка допускает одновременный захват примитива несколькими читателями, если объект не заблокирован писателем.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
rw	Указатель на RW-блокировку, которую надо захватить для чтения.
timeout	Таймаут ожидания в тиках таймера. Если таймаут не используется следует указывать значение FX_THREAD_INFINITE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_RWLOCK_OK	Успешное завершение.
FX_RWLOCK_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_RWLOCK_INVALID_OBJ	Объект не был ранее корректно проинициализирован функцией fx_rwlock_init.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания.
FX_THREAD_INVALID_TIMEOUT	Таймаут не равен значению FX_THREAD_INFINITE_TIMEOUT и не лежит в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.

Ремарки

В результате выполнения данной функции вызывающий поток может быть вытеснен.

Пример

```
fx_rwlock_t rwlock;

void my_thread(void* arg)
{
    //
    // Acquire the rwlock as a reader with 10-ticks timeout.
    //
    fx_rwlock_rd_timedlock(&rwlock, 10);
    // ...;
}
```

fx_rwlock_unlock

```
int fx_rwlock_unlock(fx_rwlock_t* rw);
```

Описание

Освобождение RW-блокировки после ее захвата в качестве читателя или писателя. Ожидающие потоки будут разблокированы в соответствии с логикой объекта.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
rw	Указатель на RW-блокировку, которую надо разблокировать.

Возвращаемое значение

Возвращаемое значение	Описание
FX_RWLOCK_OK	Успешное завершение.
FX_RWLOCK_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_RWLOCK_INVALID_OBJ	Объект не был ранее корректно проинициализирован функцией fx_rwlock_init.

Ремарки

В результате выполнения данной функции вызывающий поток может быть вытеснен.

Пример

```
fx_rwlock_t rwlock;

void my_thread(void* arg)
{
    //
    // Release the rwlock.
    //
    fx_rwlock_unlock(&rwlock);
    // ...;
}
```

fx_rwlock_wr_lock

```
int fx_rwlock_wr_lock(fx_rwlock_t* rw, fx_event_t* cancel_event);
```

Описание

Блокировка писателем с опциональным отменяющим событием. RW-блокировка допускает владение объектом только для одного писателя в каждый момент времени.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	❌
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
rw	Указатель на RW-блокировку, которую надо захватить для записи.
cancel_event	Отменяющее событие. Может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_RWLOCK_OK	Успешное завершение.
FX_RWLOCK_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_RWLOCK_INVALID_OBJ	Объект не был ранее корректно проинициализирован функцией fx_rwlock_init.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.
FX_THREAD_WAIT_CANCELLED	Ожидание было прервано из-за установки отменяющего события в активное состояние.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

В результате выполнения данной функции вызывающий поток может быть вытеснен.

Пример

```
fx_rwlock_t rwlock;
void my_thread(void* arg)
{
    //
    // Acquire the rwlock as writer.
    //
    fx_rwlock_wr_lock(&rwlock, NULL);
    // ...;
}
```

fx_rwlock_wr_timedlock

```
int fx_rwlock_wr_timedlock (fx_rwlock_t* rw, uint32_t timeout);
```

Описание

Блокировка писателем с опциональным таймаутом. RW-блокировка допускает владение объектом только для одного писателя в каждый момент времени.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
rw	Указатель на RW-блокировку, которую надо захватить для записи.
timeout	Таймаут ожидания в тиках таймера. Если таймаут не используется следует указывать значение FX_THREAD_INFINITE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_RWLOCK_OK	Успешное завершение.
FX_RWLOCK_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_RWLOCK_INVALID_OBJ	Объект не был ранее корректно проинициализирован функцией fx_rwlock_init.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания.
FX_THREAD_INVALID_TIMEOUT	Таймаут не равен значению FX_THREAD_INFINITE_TIMEOUT и не лежит в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.

Ремарки

В результате выполнения данной функции вызывающий поток может быть вытеснен.

Пример

```
fx_rwlock_t rwlock;

void my_thread(void arg)*
{
    //
    // Acquire the rwlock as a writer with 10-ticks timeout.
    //
    fx_rwlock_wr_timedlock(&rwlock, 10);
    // ...;
}
```

Барьеры (опциональный компонент)

fx_barrier_deinit

```
int fx_barrier_deinit(fx_barrier_t* barrier);
```

Описание

Удаление барьера. Все потоки, ожидающие барьер, возобновляют выполнение со статусом FX_THREAD_WAIT_DELETED. После вызова этой функции барьер не должен использоваться.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
barr	Указатель на барьер, который надо удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BARR_OK	Успешное завершение.
FX_BARR_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_barrier_init.
FX_BARR_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_barrier_t barrier;

void my_thread(void* arg)
{
    //
    // Delete the barrier.
    //
    fx_barrier_deinit(&barrier);
}
```

fx_barrier_init

```
int fx_barrier_init(fx_barrier_t* barr, unsigned int num);
```

Описание

Инициализация барьера.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
barr	Указатель на барьер, который надо проинициализировать.
num	Количество потоков, которые должны ожидать барьер для его активации.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BARR_OK	Успешное завершение.
FX_BARR_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_BARR_ZERO_LIMIT	Нулевой барьер (num == 0).

Ремарки

Нет.

Пример

```
fx_barrier_t barrier;

void my_thread(void* arg)
{
    //
    // Initialize the barrier (5 threads).
    //
    fx_barrier_init(&barrier, 5);
}
```

fx_barrier_timedwait

```
int fx_barrier_timedwait(fx_barrier_t* barr, fx_barrier_key_t* key, uint32_t timeout);
```

Описание

Ожидание барьера с таймаутом. Если данный поток (вызвавший функцию ожидания барьера) является N-ным по счету, где N – число, указанное как максимальное количество ждущих потоков при инициализации барьера, тогда все N потоков, ждущих барьера, возобновят выполнение. Поток, которая вызвал разблокирование барьера, получает специальное значение в качестве ключа (FX_BARRIER_SERIAL_THREAD), для остальных потоков значение по адресу key не меняется.
Для бесконечного ожидания, в качестве таймаута следует использовать значение FX_THREAD_INFINITE_TIMEOUT.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
barr	Указатель на барьер.
key	Переменная, в которой будет сохранено значение FX_BARRIER_SERIAL_THREAD для потока, который разблокирует барьер. Опциональный аргумент, может быть NULL.
timeout	Таймаут в тиках таймера (обычно 1 тик = 1 или 10 мс). Если в качестве таймаута указано 0 – функция возвращается немедленно.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BARR_OK	Успешное завершение.
FX_BARR_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_barrier_init.
FX_BARR_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_INVALID_TIMEOUT	Значение таймаута не лежит в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_THREAD_WAIT_TIMEOUT	Ожидание было отменено из-за таймаута.
FX_THREAD_WAIT_DELETED	Барьер был удален во время ожидания.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.
Выход из функции ожидания по событиям не связанным с барьером не влечет изменения состояния последнего. То есть если, например, в качестве максимального количества ждущих потоков было указано 2, после чего один поток вызвал fx_barrier_timedwait, но ожидание было прервано в связи с таймаутом, то следующий поток, вызвавшая fx_barrier_timedwait, будет разблокирован, хотя фактически барьер будет ожидать только он.

Пример

```
fx_barrier_t barrier;

void my_thread(void* arg)
{
    fx_barrier_key_t key = 0;
    //
    // Wait the barrier with 10 ticks timeout.
    //
    fx_barrier_timedwait(&barrier, &key, 10);
}
```

fx_barrier_wait

```
int fx_barrier_wait(fx_barrier_t* barr, fx_barrier_key_t* key, fx_event_t* cancel_event);
```

Описание

Ожидание барьера с отменяющим событием. Если данний поток (вызвавший функцию ожидания барьера) является N-ным по счету, где N – число, указанное как максимальное количество ждущих потоков при инициализации барьера, тогда все N потоков, ждущих барьера, возобновят выполнение. Поток, который вызвал разблокирование барьера, получает специальное значение в качестве ключа (FX_BARRIER_SERIAL_THREAD), для остальных потоков значение по адресу key не меняется.
При ожидании барьера может быть также указано отменяющее событие. Если во время ожидания барьера это событие будет установлено в активное состояние, заблокированный поток продолжит выполнение.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
barr	Указатель на барьер.
key	Переменная, в которой будет сохранено значение FX_BARRIER_SERIAL_THREAD для потока, который разблокирует барьер. Опциональный аргумент, может быть NULL.
cancel_event	Отменяющее событие, прерывает ожидание барьера будучи установленным в активное состояние.

Возвращаемое значение

Возвращаемое значение	Описание
FX_BARR_OK	Успешное завершение.
FX_BARR_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_barrier_init.
FX_BARR_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_CANCELLED	Ожидание было отменено из-за установки отменяющего события в активное состояние.
FX_THREAD_WAIT_DELETED	Один из объектов (барьер, или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.
Важно отметить, что выход из функции ожидания по событиям, не связанным с барьером не влечет изменения состояния последнего. То есть если, например, в качестве максимального количества ждущих потоков было указано 2, после чего один поток вызвал fx_barrier_wait, но ожидание было прервано отменяющим событием, то следующий поток, вызвавший fx_barrier_wait будет разблокирован, хотя фактически барьер будет ожидать только он.

Пример

```
fx_barrier_t barrier;
fx_event_t cancel_event;

void my_thread(void* arg)
{
    fx_barrier_key_t key = 0;
    //
    // Wait for the barrier.
    //
    fx_barrier_wait(&barrier, &key, &cancel_event);
}
```

Мьютексы

fx_mutex_acquire

```
int fx_mutex_acquire(fx_mutex_t* mutex, fx_event_t* cancel_event);
```

Описание

Захват мьютекса с отменяющим событием. Если используется потолок приоритета (если он был указан при инициализации мьютекса) приоритет потока-владельца мьютекса повышается до указанного уровня на время владения мьютексом.
Отменяющее событие можно использовать для отмены ожидания.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mutex	Указатель на мьютекс, который надо захватить.
cancel_event	Отменяющее событие. Может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MUTEX_OK	Успешное завершение. Мьютекс захвачен данной потоком.
FX_MUTEX_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_mutex_init.
FX_MUTEX_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_DELETED	Один из объектов (мьютекс или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_THREAD_WAIT_CANCELLED	Отменяющее событие было установлено в активное состояние во время ожидания. Мьютекс не захвачен.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_mutex_t mutex;

void my_thread(void* arg)
{
    //
    // Acquire the mutex.
    //
    fx_mutex_acquire(&mutex, NULL);
    // ...;
}
```

fx_mutex_deinit

```
int fx_mutex_deinit(fx_mutex_t* mutex);
```

Описание

Удаление мьютекса. Все ожидающие потоки возобновляют выполнение со статусом FX_THREAD_WAIT_DELETED. После вызова этой функции мьютекс не должен использоваться.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mutex	Указатель на мьютекс, который надо удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MUTEX_OK	Успешное завершение.
FX_MUTEX_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_mutex_init.
FX_MUTEX_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.
Если в момент удаления мьютекса у него был владелец (мьютекс был захвачен) и мьютекс был инициализирован как мьютекс с потолком приоритета, приоритет владельца не будет понижен.

Пример

```
fx_mutex_t mutex;

void my_thread(void* arg)
{
    //
    // Delete the mutex.
    //
    fx_mutex_deinit(&mutex);
    // ...;
}
```

fx_mutex_get_owner

```
fx_thread_t* fx_mutex_get_owner(fx_mutex_t* mutex);
```

Описание

Получение владельца мьютекса.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	❌
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
mutex	Указатель на мьютекс, владельца которого необходимо получить.

Возвращаемое значение

Возвращаемое значение	Описание
Указатель на поток-владелец мьютекса.	Успешное завершение.
NULL	Мьютекс свободен (нет владельца), либо объект не был корректно инициализирован с помощью функции fx_mutex_init.

Ремарки

Нет.

Пример

```
fx_mutex_t mutex;

void my_thread(void* arg)
{
    fx_thread_t* owner;
    owner = fx_mutex_get_owner(&mutex);
    // ...;
}
```

fx_mutex_init

```
int fx_mutex_init(fx_mutex_t* mutex, unsigned int ceiling, const fx_sync_policy_t policy);
```

Описание

Инициализация мьютекса.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
mutex	Указатель на мьютекс, который надо проинициализировать.
ceiling	Потолок приоритета для мьютекса. Поток, владеющий мьютексом получает этот приоритет на время владения (он должен быть равен максимальному приоритету среди приоритетов потоков, которые могут владеть мьютексом. Если потолок не используется для данного мьютекса, следует оказывать значение FX_MUTEX_CEILING_DISABLED.
policy	Политика уведомления ожидающих потоков. Определяется используемым слоем синхронизации. По умолчанию должно использоваться значение FX_SYNC_POLICY_DEFAULT. В случае явного указания политики могут поддерживаться следующие значения: 1. FX_SYNC_POLICY_FIFO – уведомление ожидающих потоков в том порядке, в котором они начали ожидание. 2. FX_SYNC_POLICY_PRIO – уведомление в первую очередь ожидателей с максимальным приоритетом. Так как поддержка различных политик зависит от конфигурации ОС, рекомендуется использовать значение по-умолчанию в тех случаях, когда требуется достичь максимальной переносимости между различными конфигурациями.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MUTEX_OK	Успешное завершение.
FX_MUTEX_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_MUTEX_INVALID_PRIO	Некорректное значение приоритета для потолка мьютекса (значение не равно FX_MUTEX_CEILING_DISABLED и не находится в диапазоне разрешенных приоритетов для потоков от 0 до FX_SCHED_ALG_PRIO_IDLE-1).
FX_MUTEX_UNSUPPORTED_POLICY	Некорректное значение аргумента policy.

Ремарки

Нет.

Пример

```
fx_mutex_t mutex1;
fx_mutex_t mutex2;

void my_thread(void* arg)
{
    //
    // Initialize the mutex1 with priority ceiling 0.
    //
    fx_mutex_init(&mutex1, 0, FX_SYNC_POLICY_FIFO);
    //
    // Initialize the mutex2 with priority ceiling disabled.
    //
    fx_mutex_init(&mutex2, FX_MUTEX_CEILING_DISABLED, FX_SYNC_POLICY_FIFO);
    // ...;
}
```

fx_mutex_release

```
int fx_mutex_release(fx_mutex_t* mutex);
```

Описание

Освобождение мьютекса. Если используется потолок приоритета (если он был указан при инициализации мьютекса) приоритет потока-владельца мьютекса понижается до начального уровня. Если у мьютекса есть ожидатели, одна поток из ожидающих мьютекс возобновит выполнение.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	❌
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
mutex	Указатель на освобождаемый мьютекс.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MUTEX_OK	Успешное завершение.
FX_MUTEX_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_mutex_init.
FX_MUTEX_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_MUTEX_WRONG_OWNER	Поток, который пытается освободить мьютекс, не является его владельцем.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_mutex_t mutex;

void my_thread(void* arg)
{
    //
    // Releasing the mutex.
    //
    fx_mutex_release(&mutex);
    // ...;
}
```

fx_mutex_timedacquire

```
int fx_mutex_timedacquire(fx_mutex_t* mutex, uint32_t timeout);
```

Описание

Захват мьютекса с таймаутом. Если используется потолок приоритета (если он был указан при инициализации мьютекса) приоритет потока-владельца мьютекса повышается до указанного уровня на время владения мьютексом.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mutex	Указатель на мьютекс, который надо захватить.
timeout	Таймаут ожидания в тиках таймера. Для ожидания с бесконечным таймаутом следует использовать значение FX_THREAD_INFINITE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MUTEX_OK	Успешное завершение. Мьютекс захвачен данной потоком.
FX_MUTEX_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_mutex_init.
FX_MUTEX_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_INVALID_TIMEOUT	Таймаут не равен FX_THREAD_INFINITE_TIMEOUT и не находится в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания. Мьютекс не захвачен.
FX_MUTEX_RECURSIVE_LIMIT	Достигнут максимальный уровень вложений для захвата мьютекса одним потоком. Счетчик вложений не изменился.
FX_MUTEX_ABANDONED	Поток-владелец завершился до освобождения мьютекса. Данные, защищаемые мьютексом, могут быть в несогласованном состоянии.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_mutex_t mutex;

void my_thread(void* arg)
{
    //
    // Acquire the mutex with timeout (10 ticks).
    //
    fx_mutex_timedacquire(&mutex, 10);
    // ...
}
```

Очереди сообщений

fx_msgq_back_send

```
int fx_msgq_back_send(fx_msgq_t* mq, uintptr_t msg, fx_event_t* cancel_event);
```

Описание

Добавление сообщения в конец очереди с отменяющим событием. Если очередь заполнена, поток, вызвавший данную функцию блокируется, пока в очереди не появится место для его сообщения (пока другой поток не вызовет fx_msgq_flush либо fx_msgq_receive).
Отменяющее событие может использоваться для прерывания ожидания.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений.
msg	Сообщение для добавления в очередь.
cancel_event	Отменяющее событие. Может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение. Сообщение было добавлено в очередь.
FX_MSGQ_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_msgq_init.
FX_MSGQ_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_CANCELLED	Ожидание было отменено из-за установки отменяющего события в активное состояние. Сообщение не добавлено в очередь.
FX_THREAD_WAIT_DELETED	Один из объектов (очередь сообщений или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_msgq_t msgq;
void my_thread(void* arg)
{
   //
   // Adding 0x11223344 at tail of the queue.
   //
   fx_msgq_back_send (&msgq, 0x11223344, NULL);
}
```

fx_msgq_back_timedsend

```
int fx_msgq_back_timedsend(fx_msgq_t* mq, uintptr_t msg, uint32_t timeout);
```

Описание

Добавление сообщения в конец очереди с таймаутом. Если очередь заполнена, поток, вызвавший данную функцию блокируется, пока в очереди не появится место для его сообщения (пока другой поток не вызовет fx_msgq_flush либо fx_msgq_receive), либо пока не истечет таймаут ожидания.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений.
msg	Сообщение для добавления в очередь.
timeout	Таймаут ожидания в тиках таймера. Для ожидания с бесконечным таймаутом следует использовать значение FX_THREAD_INFINITE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение. Сообщение было добавлено в очередь.
FX_MSGQ_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_msgq_init.
FX_MSGQ_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания.
FX_THREAD_INVALID_TIMEOUT	Таймаут не равен FX_THREAD_INFINITE_TIMEOUT и не лежит в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_msgq_t msgq;

void my_thread(void* arg)
{
    //
    // Adding 0x11223344 at tail of the queue. Timeout = 10 ticks.
    //
    fx_msgq_back_timedsend(&msgq, 0x11223344, 10);
}
```

fx_msgq_deinit

```
int fx_msgq_deinit(fx_msgq_t* mq);
```

Описание

Удаление очереди сообщений. Все потоки, ожидающие отправки или приёма сообщений, возобновляют выполнение со статусом FX_THREAD_WAIT_DELETED. После вызова этой функции очередь сообщений не должна использоваться.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений, которую надо удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение.
FX_MSGQ _INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_msgq_init.
FX_MSGQ _INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_msgq_t msgq;

void my_thread(void* arg)
{
    //
    // Delete the message queue.
    //
    fx_msgq_deinit(&msgq);
}
```

fx_msgq_flush

```
int fx_msgq_flush(fx_msgq_t* mq);
```

Описание

Очистка очереди сообщений. Все сообщения находящиеся в очереди удаляются. Если очередь была заполнена и на очереди сообщений были заблокированы потоки, ожидающие освобождения места в очереди, эти потоки будут разблокированы и продолжат выполнение (разблокируются только потоки, для сообщений которых имеется место в очереди, если заблокированных потоков больше, то, после заполнения очереди, оставшиеся потоки останутся заблокированными).

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений, которую надо очистить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение.
FX_MSGQ _INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_msgq_init.
FX_MSGQ _INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_msgq_t msgq;

void my_thread(void* arg)
{
    //
    // Flushing the message queue.
    //
    fx_msgq_flush(&msgq);
}
```

fx_msgq_front_send

```
int fx_msgq_front_send(fx_msgq_t* mq, uintptr_t msg, fx_event_t* cancel_event);
```

Описание

Добавление сообщения в начало очереди с отменяющим событием. Если очередь заполнена, поток, вызвавший данную функцию блокируется, пока в очереди не появится место для его сообщения (пока другой поток не вызовет fx_msgq_flush либо fx_msgq_receive). Отменяющее событие может использоваться для прерывания ожидания.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	❌
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений.
msg	Сообщение для добавления в очередь.
cancel_event	Отменяющее событие. Может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение. Сообщение было добавлено в очередь.
FX_MSGQ_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_msgq_init.
FX_MSGQ_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_CANCELLED	Ожидание было отменено из-за установки отменяющего события в активное состояние. Сообщение не добавлено в очередь.
FX_THREAD_WAIT_DELETED	Один из объектов (очередь сообщений или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_msgq_t msgq;

void my_thread(void* arg)
{
    //
    // Adding 0x11223344 at head of the queue.
    //
    fx_msgq_front_send(&msgq, 0x11223344, NULL);
}
```

fx_msgq_front_timedsend

```
int fx_msgq_front_timedsend(fx_msgq_t* mq, uintptr_t msg, uint32_t timeout);
```

Описание

Добавление сообщения в начало очереди с таймаутом. Если очередь заполнена, поток, вызвавший данную функцию блокируется, пока в очереди не появится место для его сообщения (пока другой поток не вызовет fx_msgq_flush либо fx_msgq_receive), либо пока не истечет таймаут ожидания.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений.
msg	Сообщение для добавления в очередь.
timeout	Таймаут ожидания в тиках таймера. Для ожидания с бесконечным таймаутом следует использовать значение FX_THREAD_INFINITE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение. Сообщение было добавлено в очередь.
FX_MSGQ_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_msgq_init.
FX_MSGQ_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания.
FX_THREAD_INVALID_TIMEOUT	Таймаут не равен FX_THREAD_INFINITE_TIMEOUT и не лежит в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_msgq_t msgq;

void my_thread(void* arg)
{
    //
    // Adding 0x11223344 at head of the queue. Timeout = 10 ticks.
    //
    fx_msgq_front_timedsend(&msgq, 0x11223344, 10);
}
```

fx_msgq_init

```
int fx_msgq_init(fx_msgq_t* mq, uintptr_t* buf, unsigned int num, const fx_sync_policy_t policy);
```

Описание

Инициализация очереди сообщений. Буфер, использующийся для хранения сообщений и его размер передаются в качестве параметров. Одно сообщение всегда имеет размер равный размеру типа uintptr_t.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений, которую надо проинициализировать.
buf	Указатель на буфер для хранения сообщений.
num	Размер буфера в элементах типа uintptr_t.
policy	Политика уведомления ожидающих потоков. Определяется используемым слоем синхронизации. По умолчанию должно использоваться значение FX_SYNC_POLICY_DEFAULT. В случае явного указания политики могут поддерживаться следующие значения: 1. FX_SYNC_POLICY_FIFO – уведомление ожидающих потоков в том порядке, в котором они начали ожидание. 2. FX_SYNC_POLICY_PRIO – уведомление в первую очередь ожидателей с максимальным приоритетом. Так как поддержка различных политик зависит от конфигурации ОС, рекомендуется использовать значение по-умолчанию в тех случаях, когда требуется достичь максимальной переносимости между различными конфигурациями.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение.
FX_MSGQ_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_MSGQ_INVALID_BUF	Некорректный указатель на буфер для сообщений или нулевой размер буфера .
FX_MSGQ_UNSUPPORTED_POLICY	Некорректный параметр policy.

Ремарки

Нет.

Пример

```
fx_msgq_t cond;
#define MSGQ_SIZE 10
uintptr_t msg_buffer[MSGQ_SIZE];

void my_thread(void* arg)
{
    //
    // Initialize the message queue.
    //
    fx_msgq_init(&msgq, msg_buffer, MSGQ_SIZE, FX_SYNC_POLICY_FIFO);
}
```

fx_msgq_receive

```
int fx_msgq_receive(fx_msgq_t* mq, uintptr_t* msg, fx_event_t* cancel_event);
```

Описание

Получение сообщения из очереди сообщений. Если очередь пуста, вызывающий поток будет заблокирован до тех пор, пока в очереди не появятся сообщения.
Отменяющее событие может использоваться для прерывания ожидания.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений.
msg	Указатель на переменную, в которую будет помещено сообщение, извлеченное из очереди.
cancel_event	Отменяющее событие. Может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение. Сообщение было извлечено из очереди.
FX_MSGQ_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_msgq_init.
FX_MSGQ_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_MSGQ_INVALID_BUF	Нулевой указатель на буфер-приемник сообщения.
FX_THREAD_WAIT_CANCELLED	Ожидание было отменено из-за установки отменяющего события в активное состояние. Сообщение не извлечено из очереди.
FX_THREAD_WAIT_DELETED	Один из объектов (очередь сообщений или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_msgq_t msgq;

void my_thread(void arg)*
{
    uintptr_t msg;
    if (fx_msgq_receive(&msgq, &msg, NULL) == FX_MSGQ_OK)
    {
        // msg contains received message now.
    }
    // ...
}
```

fx_msgq_timedreceive

```
int fx_msgq_timedreceive (fx_msgq_t* mq, uintptr_t* msg, uint32_t timeout);
```

Описание

Получение сообщения из очереди сообщений с таймаутом. Если очередь пуста, вызывающий поток будет заблокирован до тех пор, пока в очереди не появятся сообщения.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
mq	Указатель на очередь сообщений.
msg	Указатель на переменную, в которую будет помещено сообщение, извлеченное из очереди.
timeout	Таймаут ожидания в тиках таймера. Для ожидания с бесконечным таймаутом следует использовать значение FX_THREAD_INFINITE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение. Сообщение было извлечено из очереди.
FX_MSGQ_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_msgq_init.
FX_MSGQ_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_MSGQ_INVALID_BUF	Нулевой указатель на буфер-приемник сообщения.
FX_THREAD_INVALID_TIMEOUT	Значение таймаута не равно FX_THREAD_INFINITE_TIMEOUT и не находится в диапазоне 0 – FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_msgq_t msgq;

void my_thread(void* arg)
{
    uintptr_t msg;
    // Waiting for a message within 10-ticks timeout
    if (fx_msgq_timedreceive(&msgq, &msg, 10) == FX_MSGQ_OK)
    {
        // msg contains received message now.
    }
    // ...
}
```

Потоки

fx_thread_deinit

```
int fx_thread_deinit(fx_thread_t* thread);
```

Описание

Удаляет поток, который ранее был создан с помощью функции fx_thread_init. Функция освобождает только внутренние ресурсы потока. Память, используемая для структуры потока, а также его стек, управляются приложением. Приложение также должно предотвратить использование удаляемого потока во время и после окончания работы данной функции.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
thread	Указатель на структуру потока, которая ранее была проинициализирована с помощью fx_thread_init.

Возвращаемое значение

Код ошибки	Описание
FX_THREAD_INVALID_PTR	Некорректный (нулевой) указатель на структуру потока.
FX_THREAD_INVALID_OBJ	Неинициализированная структура потока.
FX_THREAD_OK	Успешное завершение.

Ремарки

Вызов fx_thread_deinit для данной структуры потока допускается только один раз, результат повторных вызовов fx_thread_deinit не определен.
Допускается удаление только завершенных потоков (для которых успешно завершился вызов fx_thread_join), результат вызова fx_thread_deinit для незавершившегося потока не определен.

Пример

```
fx_thread_t my_thread_struct;

void thread_example(void)
{
    int status;
    status = fx_thread_deinit(&my_thread_struct);
}
```

fx_thread_delay_until

```
int fx_thread_delay_until( uint32_t* prev_wake, uint32_t increment);
```

Описание

Ожидание с абсолютным таймаутом.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
prev_wake	Указатель на текущее значение таймаута, обновляется при каждом вызове функции (см. пример).
increment	Инкремент таймаута, вместе с prev_wake формирует абсолютный таймаут.

Возвращаемое значение

Возвращаемое значение	Описание
FX_THREAD_OK	Успешное завершение.
FX_THREAD_INVALID_TIMEOUT	Значение таймаута не лежит в диапазоне от 0 до FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_THREAD_INVALID_PTR	Нулевой указатель в качестве prev_wake.

Ремарки

Если указанное значение таймаута находится в прошлом, функция немедленно возвращает управление, значение time_to_wake при этом обновляется.

Пример

```
fx_thread_t my_thread;

void thread_example(void* arg)
{
    //
    // Get absolute tick count.
    //
    uint32_t prev_wake = fx_timer_get_tick_count();
    while(1)
    {
        //
        // New value of prev_wake is *prev_wake + increment.
        // Every iteration increments prev_wake and
        // updates the user_supplied variable.
        //
        fx_thread_delay_until(&prev_wake, 100);
    }
}
```

fx_thread_exit

```
void fx_thread_exit(void);
```

Описание

Функция завершает выполнение потока из которого она была вызвана.
После того, как поток завершен, перед повторным использованием он должна быть деинициализирован с помощью fx_thread_deinit, после чего инициализирован заново с помощью fx_thread_init.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Нет.

Возвращаемое значение

Нет (функция не возвращает управление).

Ремарки

Пользователь должен гарантировать, что каждый поток завершается вызовом данной функции. Результат возврата из потоковой функции не определен.
Выполнение данной функции всегда приводит к перепланированию.

Пример

```
void thread_example(void* arg)
{
    //
    // Thread code
    //
    fx_thread_exit();
}
```

fx_thread_get_params

```
int fx_thread_get_params(fx_thread_t* thread, unsigned int type, unsigned int* value);
```

Описание

Функция позволяет получить различные параметры указанного потока (приоритет, квант, процессор и т.д.).

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
thread	Указатель на структуру потока, параметры которого надо получить.
type	Тип параметра, который нужно получить, допустимы следующие значения: 1. FX_THREAD_PARAM_PRIO – получение приоритета 2. FX_THREAD_PARAM_TIMESLICE – получение кванта 3. FX_THREAD_PARAM_CPU – получение маски процессоров, на которых разрешено выполнение данного потока.
value	Указатель на переменную, в которой будет сохранено значение указанного параметра.

Возвращаемое значение

Возвращаемое значение	Описание
FX_THREAD_OK	Успешное завершение.
FX_THREAD_ INVALID_OBJ	Некорректный поток (некорректно инициализированный).
FX_THREAD_INVALID_PTR	Нулевой указатель либо на поток, либо на параметр value.
FX_THREAD_INVALID_PARAM	Некорректное значение параметра type.

Ремарки

Если приоритет потока был динамически повышен (например, из-за захвата мьютекса) то данная функция вернет актуальный (повышенный) приоритет.
Значение кванта 0 означает, что кванты для данного потока не используются.

Пример

```
fx_thread_t my_thread;

void thread_example(void* arg)
{
    unsigned int prio;
    //
    // Get current priority of specified thread.
    //
    int status = fx_thread_get_params(&my_thread, FX_THREAD_PARAM_PRIO, &prio);
}
```

fx_thread_init

```
int fx_thread_init(fx_thread_t *thread, void (*func)(void*), void* arg, unsigned int priority, void* stack,
size_t stack_sz, bool create_suspended);
```

Описание

Функция создает поток, который начинает выполнение с указанной функции с заданым аргументом. С помощью параметров передается приоритет, адрес и размер стека, а также начальное состояние (работает/приостановлен).
Для установки кванта следует использовать функцию fx_thread_set_params. По умолчанию поток инициализируется с бесконечным квантом.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
thread	Указатель на структуру потока, которую следует проинициализировать.
func	Потоковая функция (должна иметь тип void (*)(void*).
arg	Аргумент потоковой функции.
priority	Приоритет создаваемого потока. Данное значение должно быть в диапазоне от 0 до значения (FX_SCHED_ALG_PRIO_IDLE – 1). Количество приоритетов определяется в результате конфигурирования. Приоритет 0 соответствует наиболее приоритетному потоку, FX_SCHED_ALG_PRIO_IDLE – 1 – наименее приоритетному.
stack	Базовый адрес стека потока.
stack_sz	Размер стека. Начальный указатель стека потока будет установлен по адресу stack + stack_sz, поэтому оба значения должны быть выровнены в соответствии с требованиями процессора.
create_suspended	Определяет, должен ли новая поток начать выполнение сразу после вызова данной функции. Если это значение равно true, поток инициализируется в приостановленном состоянии и пользователь должен дополнительно вызвать функцию fx_thread_resume, чтобы перевести поток в состояние готовности к выполнению. Если это значение равно false – новый поток становится готовым к выполнению непосредственно после вызова данной функции.

Возвращаемое значение

Код ошибки	Описание

FX_THREAD_INVALID_PTR	Некорректный (нулевой) указатель на структуру потока.
FX_THREAD_INVALID_ENTRY	Некорректная потоковая фукция.
FX_THREAD_NO_STACK	Некорректный указатель стека потока.
FX_THREAD_NO_STACK	Недостаточный размер стека потока.
FX_THREAD_INVALID_PRIO	Неверное значение приоритета.
FX_THREAD_OK	Успешное завершение.

Ремарки

Если приоритет создаваемого потока выше (численно меньшее значение приоритета), чем приоритет потока, в контексте которого вызывается функция fx_thread_init, а также значение create_suspended равно false, вызывающий поток вытесняется вновь созданным.
В процессе создания потока он добавляется в структуры планировщика, поэтому, в завимости от используемого планировщика, вызов fx_thread_init может не быть идемпотентным.
Ответственность за существование и доступность памяти, используемой потоковой структурой, а также стеком потока, лежит на приложении.

Поток начинает выполнение непосредственно с адреса, указанного в качестве аргумента функции, не используется никаких дополнительных функций-оберток, поэтому, если поток не является бесконечным циклом, он должен завершаться явно с помощью функции fx_thread_exit.

Пример

```
fx_thread_t my_thread_struct;
void my_thread_func(void* arg);
int stack[0x100];

void thread_example(void)
{
        int error = fx_thread_init(&my_thread_struct, my_thread_func, 0x11223344, 10, stack, sizeof(stack),
false);

        if(error)
        {
                // Error handling
        }
}

void my_thread_func(void* arg)
{
        assert((uintptr_t)arg == 0x11223344);
    //
        // Thread code
    //
        fx_thread_exit();
}
```

fx_thread_join

```
int fx_thread_join(fx_thread_t* thread);
```

Описание

Функция возвращает код завершения указанного потока, если этот поток завершился на момент вызова. Если указанный в качестве параметра поток не завершен, выполнение fx_thread_join приостанавливается до его завершения.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
thread	Указатель на структуру потока.

Возвращаемое значение

Код ошибки	Описание
FX_THREAD_INVALID_PTR	Некорректный (нулевой) указатель на структуру потока.
FX_THREAD_INVALID_OBJ	Неинициализированная структура потока.
FX_THREAD_JOIN_SELF	Вызов потоком данной функции для ожидания завершения самого себя.
FX_THREAD_OK	Успешное завершение.

Ремарки

Данной функцией могут пользоваться одновременно несколько потоков (ожидать завершения другого потока), если эти потоки и ожидаемый поток находятся на разных процессорах, тогда возврат из функции fx_thread_join не означает, что закончилось уведомление всех ожидающих потоков.

Пример

```
fx_thread_t my_thread;

void thread_example(void* arg)
{
    int status;
    //
    // Waiting for completion of my_thread
    //
    status = fx_thread_join(&my_thread);

    if(status == FX_THREAD_OK)
    {
        // Thread completed
    }
}
```

fx_thread_resume

```
int fx_thread_resume(fx_thread_t* thread);
```

Описание

Функция возобновляет выполнение указанного потока, который был ранее приостановлен с помощью fx_thread_suspend или создан в приостановленном состоянии с помощью fx_thread_init.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
thread	Указатель на структуру потока, который требуется возобновить.

Возвращаемое значение

Код ошибки	Описание
FX_THREAD_INVALID_PTR	Некорректный (нулевой) указатель на структуру потока.
FX_THREAD_INVALID_OBJ	Неинициализированная структура потока.
FX_THREAD_OK	Успешное завершение.

Ремарки

В многопроцессорных системах, возобновление потоков, которые выполняются на других процессорах, может происходить асинхронно.
В случае, если указанный поток находится в состоянии ожидания, вызовы fx_thread_resume не влияют на состояние потока.
Если поток находится в состоянии готовности, вызовы fx_thread_resume для этого потока игнорируются.

Пример

```
fx_thread_t my_thread;

void thread_example(void* arg)
{
    int status = fx_thread_resume(&my_thread);
}
```

fx_thread_self

```
fx_thread_t* fx_thread_self(void);
```

Описание

Функция используется для получения указателя на поток, выполняющийся в данное время (вызвавший fx_thread_self).

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Нет.

Возвращаемое значение

Указатель на текущий поток.

Ремарки

В некоторых конфигурациях функция может повышать SPL до SCHED_LEVEL, поэтому запрещается ее использование в контексте, где SPL может быть выше этого уровня, например внутри области кода с захваченной spin-блокировкой.

Пример

```
void thread_example(void* arg)
{
    fx_thread_t* me = fx_thread_self();
}
```

fx_thread_set_params

```
int fx_thread_set_params(fx_thread_t* thread, unsigned int type, unsigned int value);
```

Описание

Функция позволяет установить параметры (приоритет, квант, процессор) для указанного потока.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
thread	Указатель на структуру потока, параметры которого надо установить.
type	Тип параметра. Допустимы следующие значения: 1. FX_THREAD_PARAM_PRIO – установка приоритета 2. FX_THREAD_PARAM_TIMESLICE – установка кванта 3. FX_THREAD_PARAM_CPU – установка маски процессоров, на которых разрешено выполнение данного потока.
value	Значение параметра, которое следует установить для указанного потока.

Возвращаемое значение

Возвращаемое значение	Описание
FX_THREAD_OK	Успешное завершение.
FX_THREAD_INVALID_OBJ	Некорректный поток (некорректно проинициализированный).
FX_THREAD_INVALID_PTR	Нулевой указатель на структуру потока.
FX_THREAD_INVALID_PARAM	Некорректный параметр type.
FX_THREAD_INVALID_PRIO	Неверный приоритет (не лежащий в диапазоне от 0 до FX_SCHED_ALG_PRIO_IDLE-1.
FX_THREAD_PARAM_TIMESLICE	Некорректное значение кванта (не лежащее в диапазоне от 1 до FX_TIMER_MAX_RELATIVE_TIMEOUT).
FX_THREAD_PARAM_CPU	Некорректный номер процессора (не лежащий в диапазоне от 0 до максимального номера процессора в системе).

Ремарки

Если приоритет потока был динамически повышен (например, из-за захвата мьютекса) то установленный данной функцией приоритет будет перезаписан старым приоритетом, который будет установлен после выхода из мьютекса.
Обновление кванта не влечет перепланирования, новый квант будет использован планировщиком только при следующем перепланировании.

Обновление маски процессоров выполняемое из самого потока происходит синхронно, после выхода из функции поток будет уже на новом процессоре, если текущий не входит в новую маску. Обновление маски процессоров запрошенное из других потоков будет применено только после следующего перехода потока из состояния ожидания в состояние готовности.
Изменения всех параметров для потоков, находящихся на других процессорах, происходит асинхронно.

Пример

```
fx_thread_t my_thread;
int my_thread_stk[1024/sizeof(int)];

void thread_example(void* arg)
{
    unsigned int timeslice = 10;
    fx_sched_state_t prev;

        //
        // Create thread an set TIMESLICE parameter "atomically" with locked scheduler
        //
        fx_sched_lock(&prev);

    fx_thread_init(
            &my_thread,
            my_thread_fn,
            NULL,
            10,
            my_thread_stk,
            sizeof(my_thread_stk),
            false
        );

    //
    // Setting timeslice for round-robin scheduling for thread "my_thread".
    //
    int status = fx_thread_set_params(&my_thread, FX_THREAD_PARAM_TIMESLICE, timeslice);

        fx_sched_unlock(prev);
}
```

fx_thread_sleep

```
int fx_thread_sleep(uint32_t ticks);
```

Описание

Функция приостанавливает выполнение потока, из которого она вызвана, на указанное количество тиков таймера (количество тиков в секунду конфигурируется, обычно 1 тик равен 1 или 10 мс).

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
ticks	Длительность приостановки потока в тиках таймера. Максимальный таймаут определяется как FX_TIMER_MAX_RELATIVE_TIMEOUT, значение ticks не должно быть больше этого значения.

Возвращаемое значение

Код ошибки	Описание
FX_THREAD_INVALID_TIMEOUT	Значение таймаута превышает FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_THREAD_OK	Успешное завершение.

Ремарки

Вызов функции с нулевым аргументом приводит к немедленному возврату в вызывающую функцию с успешным кодом завершения (перепланирование потоков или обновление кванта потока при этом не производится).

Пример

```
void thread_example(void* arg)
{
    //
    // Suspend current thread for 1000 timer ticks.
    //
    fx_thread_sleep(1000);
    //
    // Control flow gets here after delay.
    //
}
```

fx_thread_suspend

```
int fx_thread_suspend(void);
```

Описание

Функция приостанавливает выполнение текущего потока.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Нет.

Возвращаемое значение

Код ошибки	Описание
FX_THREAD_OK	Успешное завершение.

Ремарки

Поток возобновляет выполнение в случае вызова fx_thread_resume другим потоком (с аргументом соответствующим текущему потоку).

Пример

```
void thread_example(void* arg)
{
    //
    // Suspend thread execution
    //
    fx_thread_suspend();
    //
    // The thread gets here after successful call of fx_thread_resume.
    //
}
```

fx_thread_terminate

```
int fx_thread_terminate(fx_thread_t* thread);
```

Описание

Функция останавливает выполнение потока независимо от его текущего статуса.
После того, как поток завершен, перед повторным использованием он должна быть деинициализирован с помощью fx_thread_deinit, после чего инициализирован заново с помощью fx_thread_init.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	❌
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
thread	Указатель на структуру потока, выполнение которого следует прервать.

Возвращаемое значение

Код ошибки	Описание
FX_THREAD_INVALID_PTR	Некорректный (нулевой) указатель на структуру потока.
FX_THREAD_INVALID_OBJ	Неинициализированная структура потока.
FX_THREAD_OK	Успешное завершение.

Ремарки

Если выполнение потока включает побочные эффекты (выделение памяти, захват ресурсов и т.д.), то отмена всех побочных эффектов не гарантируется, использование этой функции потенциально может приводить к ошибкам и некорректному использованию ресурсов, поэтому функция не предназначена для использования в прикладном коде и должна использоваться только для целей отладки. Для завершения потоком самого себя следует использовать функцию fx_thread_exit.
В многопроцессорной системе завершение потока выполняется асинхронно, если удаляемый поток выполняется на другом процессоре, для ожидания его фактического завершения следует использовать fx_thread_join.
Если завершаемый поток находится на том же процессоре, на котором вызывается данная функция, это может привести к перепланированию потоков на данном процессоре.

Пример

```
fx_thread_t my_thread_struct;

void thread_example(void)
{
    int status;
    //
    // Terminate thread associated with my_thread_struct.
    //
    status = fx_thread_terminate(&my_thread_struct);

    if(status == FX_THREAD_OK)
    {
        //
        // Wait for thread termination.
        //
        status = fx_thread_join(&my_thread_struct);
    }
}
```

fx_thread_timedwait_event

```
int fx_thread_timedwait_event(fx_event_t event,* uint32_t timeout);
```

Описание

Ожидание объекта «событие» с таймаутом.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
event	Указатель на объект «событие», которого следует ожидать.
timeout	Таймаут ожидания. Если таймаут равен 0, тогда функция проверяет состояние события и если оно не установлено в активное состояние, немедленно возвращается с результатом FX_THREAD_WAIT_TIMEOUT. Для ожидания с бесконечным таймаутом следует использовать значение FX_THREAD_INFINITE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_THREAD_OK	Успешное завершение (объект событие был установлен в активное состояние).
FX_THREAD_WAIT_DELETED	Объект стал некорректным во время ожидания (был удален).
FX_THREAD_WAIT_TIMEOUT	Событие не было установлено в течении промежутка времени, указанного в качестве таймаута.
FX_EVENT_INVALID_OBJ	Объект "событие" не был корректно проинициализирован.

Ремарки

Нет.

Пример

```
fx_event_t event;

void my_thread(void* arg)
{
    //
    // This thread will be suspended here until the event is set
    // or timeout (10 ticks) is exceeded.
    //
    int status = fx_thread_timedwait_event(&event, 10);
}
```

fx_thread_wait_event

```
int fx_thread_wait_event( fx_event_t* event, fx_event_t* cancel_event);
```

Описание

Ожидание объекта «событие». Для отмены ожидания может использоваться дополнительный объект «событие». Выполнение вызвавшего потока приостанавливается пока не будет установлено одно из этих событий.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
event	Указатель на объект «событие».
cancel_event	Указатель на объект «событие» использующийся для отмены ожидания основного объекта. Если отменяющее событие не используется этот параметр может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_THREAD_OK	Успешное завершение (основной объект событие был установлен в активное состояние).
FX_THREAD_WAIT_DELETED	Один из объектов стал некорректным во время ожидания (был удален).
FX_THREAD_WAIT_CANCELLED	Отменяющее событие было установлено в активное состояние.
FX_EVENT_INVALID_OBJ	Основной объект не был корректно проинициализирован, либо отменяющее событие было указано (отлично от NULL), но не является корректным объектом.

Ремарки

Нет.

Пример

```
fx_event_t event;

void thread_example(void* arg)
{
    //
    // This thread will be suspended here until the event is set.
    //
    int status = fx_thread_wait_event(&event, NULL);
}
```

fx_thread_yield

```
void fx_thread_yield(void);
```

Описание

Функция используется для передачи другому потоку (с тем же приоритетом) оставшегося кванта планировщика текущего потока.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Нет.

Возвращаемое значение

Нет.

Ремарки

Если текущий используемый планировщик не поддерживает кванты (например, позволяет иметь только один активный поток на каждый приоритет) fx_thread_yield не производит никаких действий.

Пример

```
void thread_example(void* arg)
{
    fx_thread_yield();
}
```

Семафоры

fx_sem_deinit

```
int fx_sem_deinit(fx_sem_t* sem);
```

Описание

Удаление семафора. Все ожидающие потоки будут разблокированы с кодом ошибки FX_THREAD_WAIT_DELETED.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
sem	Указатель на семафор, который надо удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM_ INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_SEM_ INVALID_OBJ	Объект не был корректно проинициализирован с помощью функции fx_sem_init.

Ремарки

Нет.

Пример

```
fx_sem_t sem;

void my_thread(void* arg)
{
    //
    // Delete the semaphore.
    //
    fx_sem_deinit(&sem);
}
```

fx_sem_get_value

```
int fx_sem_get_value(fx_sem_t* sem, unsigned int* value);
```

Описание

Получение текущего значения счетчика семафора.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
sem	Указатель на семафор, счетчик которого необходимо получить.
value	Указатель на переменную, в которую следует сохранить текущее значение счетчика семафора.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM_ INVALID_PTR	Некорректный (нулевой) указатель на объект или на переменную value.
FX_SEM_ INVALID_OBJ	Объект не был корректно проинициализирован с помощью функции fx_sem_init.

Ремарки

Нет.

Пример

```
fx_sem_t sem;

void my_thread(void* arg)
{
    unsigned semaphore;
    fx_sem_get_value(&sem, &semaphore);
}
```

fx_sem_init

```
int fx_sem_init(fx_sem_t* sem, unsigned int init, unsigned int max, const fx_sync_policy_t policy);
```

Описание

Инициализация семафора. Начальное и максимальное значение указывается в качестве параметров.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
sem	Указатель на семафор, который надо проинициализировать.
init	Начальное значение семафора.
max	Максимальное значение семафора.
policy	Политика уведомления ожидающих потоков. Определяется используемым слоем синхронизации. По умолчанию должно использоваться значение FX_SYNC_POLICY_DEFAULT. В случае явного указания политики могут поддерживаться следующие значения: 1. FX_SYNC_POLICY_FIFO – уведомление ожидающих потоков в том порядке, в котором они начали ожидание. 2. FX_SYNC_POLICY_PRIO – уведомление в первую очередь ожидателей с максимальным приоритетом. Так как поддержка различных политик зависит от конфигурации ОС, рекомендуется использовать значение по-умолчанию в тех случаях, когда требуется достичь максимальной переносимости между различными конфигурациями.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM_ INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_SEM_UNSUPPORTED_POLICY	Некорректное значение параметра policy.

Ремарки

Нет.

Пример

```
fx_sem_t sem;

void my_thread(void* arg)
{
    //
    // Initialize binary semaphore.
    //
    fx_sem_init(&sem, 0, 1, FX_SYNC_POLICY_FIFO);
}
```

fx_sem_post

```
int fx_sem_post(fx_sem_t* sem);
```

Описание

Увеличение счетчика семафора на 1. Если есть ожидающие потоки, один из них будет разблокирован и продолжит выполнение.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
sem	Указатель на семафор, счетчик которого необходимо увеличить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM_ INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_SEM_ INVALID_OBJ	Объект не был корректно проинициализирован с помощью функции fx_sem_init.

Ремарки

Вызывающий поток может быть вытеснен.

Пример

```
fx_sem_t sem;

void my_thread(void* arg)
{
    fx_sem_post(&sem);
}
```

fx_sem_reset

```
int fx_sem_reset(fx_sem_t* sem);
```

Описание

Сброс значения счетчика семафора в 0.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
sem	Указатель на семафор, который требуется сбросить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM_ INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_SEM_ INVALID_OBJ	Объект не был корректно проинициализирован с помощью функции fx_sem_init.

Ремарки

Нет.

Пример

```
fx_sem_t sem;

void my_thread(void* arg)
{
    //
    // Reset the semaphore.
    //
    fx_sem_reset(&sem);
}
```

fx_sem_timedwait

```
int fx_sem_timedwait(fx_sem_t* sem, uint32_t timeout);
```

Описание

Уменьшение счетчика семафора на 1. Если счетчик равен 0 в момент вызова функции, вызывающий поток блокируется до тех пор, пока счетчик семафора не будет увеличен с помощью функции fx_sem_post.
Для отмены ожидания может дополнительно указываться таймаут.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
sem	Указатель на семафор, который надо уменьшить.
timeout	Таймаут ожидания в тиках таймера (1 тик обычно равен 1 или 10 мс). Для бесконечного таймаута следует использовать значение FX_THREAD_INFINITE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM _INVALID_OBJ	Объект не был корректно проинициализирован с помощью fx_sem_init.
FX_SEM _INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_DELETED	Семафор (или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания.
FX_THREAD_INVALID_TIMEOUT	Значение таймаута не равно FX_THREAD_INFINITE_TIMEOUT и не находится в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.

Ремарки

Вызывающий поток может быть вытеснен.

Пример

```
fx_sem_t sem;

void my_thread(void* arg)
{
    fx_sem_timedwait(&sem, 10);

}
```

fx_sem_wait

```
int fx_sem_wait(fx_sem_t* sem, fx_event_t* cancel_event);
```

Описание

Уменьшение счетчика семафора на 1, если счетчик равен 0 в момент вызова функции, вызывающий поток блокируется до тех пор, пока счетчик семафора не будет увеличен с помощью функции fx_sem_post.
Для отмены ожидания может использоваться опциональное отменяющее событие.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
sem	Указатель на семафор, который надо уменьшить.
cancel_event	Отменяющее событие, может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM _INVALID_OBJ	Объект не был корректно проинициализирован с помощью fx_sem_init.
FX_SEM _INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_DELETED	Семафор (или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_THREAD_WAIT_CANCELLED	Ожидание было прерывано из-за установки отменяющего события в активное состояние.
FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

Вызывающий поток может быть вытеснен.

Пример

```
fx_sem_t sem;

void my_thread(void* arg)
{
    fx_sem_wait(&sem, NULL);
}
```

События

fx_event_deinit

```
int fx_event_deinit(fx_event_t* event);
```

Описание

Удаление объекта «событие». Все ожидатели данного события возобновляют выполнение со статусом FX_THREAD_WAIT_DELETED. Приложение должно предотвратить использование данного объекта после выполнения функции.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
event	Указатель на объект «событие», который следует удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_EVENT_OK	Успешное завершение (объект «событие» был успешно удален).
FX_EVENT_INVALID_OBJ	Объект не был ранее корректно проинициализирован с помощью функции fx_event_init.
FX_EVENT_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_event_t event;
void my_thread(void* arg)
{
        //
        // Delete the event.
        //
        fx_event_deinit(&event);
}
```

fx_event_get_state

```
int fx_event_get_state(fx_event_t* event, bool* state);
```

Описание

Получение состояние объекта «событие».

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
event	Указатель на объект «событие», состояние которого следует получить.
state	Указатель на переменную типа bool, в которую будет сохранено состояние объекта.

Возвращаемое значение

Возвращаемое значение	Описание
FX_EVENT_OK	Успешное завершение .
FX_EVENT_INVALID_OBJ	Объект не был ранее корректно проинициализирован с помощью функции fx_event_init.
FX_EVENT_INVALID_PTR	Некорректный (нулевой) указатель на объект или нулевой указатель на переменную state.

Ремарки

Нет.

Пример

```
fx_event_t event;
void my_thread(void arg)*
{
        bool state;

        //
        // Getting event state.
        //
        if(fx_event_get_state(&event, &state) == FX_EVENT_OK)
        {
        ...
        }
}
```

fx_event_init

```
int fx_event_init(fx_event_t* event, bool initial_state);
```

Описание

Инициализация объекта «событие». Память под объект предоставляется пользователем, начальное состояние указывается в качестве параметра.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
event	Указатель на объект «событие», который следует инициализировать.
initial_state	Начальное состояние события: true – если объект в активном состоянии, иначе false.

Возвращаемое значение

Возвращаемое значение	Описание
FX_EVENT_OK	Успешное завершение (объект событие был успешно инициализирован).
FX_EVENT_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

Нет.

Пример

```
fx_event_t event;
void my_thread(void* arg)
{
        //
        // Initialize event in active state.
        //
        fx_event_init(&event, true);
}
```

fx_event_reset

```
int fx_event_reset(fx_event_t* event);
```

Описание

Сброс состояния объекта «событие» в неактивное.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
event	Указатель на объект «событие», который следует сбросить в неактивное состояние.

Возвращаемое значение

Возвращаемое значение	Описание
FX_EVENT_OK	Успешное завершение .
FX_EVENT_INVALID_OBJ	Объект не был ранее корректно проинициализирован с помощью функции fx_event_init.
FX_EVENT_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

Нет.

Пример

```
fx_event_t event;
void my_thread(void* arg)
{
        //
        // Reset the event.
        //
        fx_event_reset(&event);
}
```

fx_event_set

```
int fx_event_set(fx_event_t* event);
```

Описание

Установка объекта «событие» в активное состояние. Все ожидающие потоки оповещаются и возобновляют выполнение.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
event	Указатель на объект «событие», который следует установить в активное состояние.

Возвращаемое значение

Возвращаемое значение	Описание
FX_EVENT_OK	Успешное завершение.
FX_EVENT_INVALID_OBJ	Объект не был ранее корректно проинициализирован с помощью функции fx_event_init.
FX_EVENT_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен. Состояние события может быть сброшено в неактивное только с помощью функции fx_event_reset. Оповещаемые потоки не оказывают влияния на состояние объекта.

Пример

```
fx_event_t event;
void my_thread(void* arg)
{
        //
        // Set the event. All waiters will be notified.
        //
        fx_event_set(&event);
}
```

Таймеры

fx_timer_cancel

```
int fx_timer_cancel(fx_timer_t* timer);
```

Описание

Отмена ранее установленного таймера. Если таймер на момент вызова функции еще не был активирован, он будет отменен. Если таймер был неактивен – не будет произведено никаких действий.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
timer	Указатель на таймер, который следует отмепоток.

Возвращаемое значение

Возвращаемое значение	Описание
FX_TIMER_OK	Успешное завершение.
FX_TIMER _INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_TIMER_INVALID_OBJ	Некорректно проинициализированный объект таймера.
FX_TIMER_ALREADY_CANCELLED	Таймер на момент вызова функции был в неактивном состоянии.

Ремарки

Функция предотвращает последующую активацию указанного таймера, а также гарантирует, что активация не происходит в текущий момент. Все используемые обработчиком ресурсы могут быть освобождены после завершения работы данной функции.

Пример

```
fx_timer_t timer;

void my_thread(void* arg)
{
        //
        // Cancel the timer.
        //
        fx_timer_cancel(&timer);
        ...;
}
```

fx_timer_deinit

```
int fx_timer_deinit(fx_timer_t* timer);
```

Описание

Удаление таймера.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
timer	Указатель на таймер, который надо удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_TIMER_OK	Успешное завершение.
FX_TIMER _INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_TIMER _INVALID_OBJ	Некорректно проинициализированный объект таймера.

Ремарки

Вызывающий поток может быть вытеснен.
Пользователь должен гарантировать, что таймер не будет использоваться во время и после работы данной функции. После отмены таймера с помощью функции fx_timer_cancel, гарантируется, что не произойдет новых активаций, а также, что функция обратного вызова не работает на каком-либо из процессоров.

Пример

```
fx_timer_t timer;

void my_thread(void* arg)
{
        //
        // Delete the timer.
        //
        fx_timer_deinit(&timer);
        ...;
}
```

fx_timer_get_tick_count

`uint32_t fx_timer_get_tick_count(void);`

Описание

Получение текущего значения системного счетчика тиков.

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Нет.

Возвращаемое значение

Текущее значение счетчика тиков.

Ремарки

Нет.

Пример

```
void my_thread(void* arg)
{
        uint32_t ticks_from_startup = fx_timer_get_tick_count();
        ...;
}
```

fx_timer_init

```
int fx_timer_init(fx_timer_t* timer, int (*func)(void*), void* arg);
```

Описание

Инициализация таймера с указанием пользовательской функции обратного вызова, которая должна вызываться в результате активации таймера. Аргумент этой функции также передается в качестве параметра.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
timer	Указатель на таймер, который надо проинициализировать.
func	Функция обратного вызова (callback), которая будет вызвана при активации таймера. Если таймер периодический, функция будет вызываться при каждой его активации.
arg	Аргумент, который будет передаваться в функцию обратного вызова при ее вызове.

Возвращаемое значение

Возвращаемое значение	Описание
FX_TIMER_OK	Успешное завершение.
FX_TIMER _INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_TIMER_INVALID_CALLBACK	Некорректный (нулево) указатель на функцию обратного вызова.

Ремарки

Аргумент, возвращаемый пользовательской функцией в данной версии API игнорируется. Такой прототип используется для возможности вызова по таймеру стандартных функций активации примитивов синхронизации, которые возвращают int и принимают указатель на объект (например, fx_sem_post, fx_event_set и пр.).
Окружение функции обратного вызова может различаться в зависимости от конфигурации (от ISR до DPC), поэтому для максимальной совестимости, следует предполагать его выполнение в режиме максимальных ограничений (в контексте ISR).

Пример

```
fx_timer_t timer;
fx_timer_t tim_event;

int my_timer_callback(void* argument)
{
        // Argument is 0x11223344 here.
        ...;
}

void my_thread(void arg)*
{
        //
        // Initialize the timer.
        //
        fx_timer_init(&timer, my_timer_callback, (void*) 0x11223344);
        ...;
}
```

fx_timer_set_abs

```
int fx_timer_set_abs(fx_timer_t* timer, uint32_t absolute_start, uint32_t period);
```

Описание

Установка времени активации таймера. Время активации указывается в абсолютной форме (относительно старта системы либо последнего вызова fx_timer_set_tick_count). Если таймер был активен, он неявно отменяется.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
timer	Указатель на таймер.
absolute_start	Абсолютное время первого срабатывания таймера. Если это значение меньше либо равно текущему значению счетчика тиков – процедура активации таймера запускается немедленно (в зависимости от реализации допускается от 0 до 1 тика задержки перед активацией).
period	Период последующих (2-го и далее) срабатываний таймера. Если это значение равно 0, таймер будет активирован один раз.

Возвращаемое значение

Возвращаемое значение	Описание
FX_TIMER_OK	Успешное завершение.
FX_TIMER _INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_TIMER_INVALID_OBJ	Некорректно проинициализированный объект таймера.
FX_TIMER_INVALID_TIMEOUT	Значение периода не находится в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_TIMER_CONCURRENT_USE	Попытка одновременно установить один и тот же таймер с разных процессоров (неопределенное поведение). Таймер не был установлен.

Ремарки

В зависимости от реализации таймеров, данная функция может иметь задержку O(N), зависящую от количества таймеров в системе. В стандартных конфигурациях функция выполняется за фиксированное время.

Пример

```
fx_timer_t timer;

void my_thread(void* arg)
{
        //
        // Timer will expire when tick count reaches 5000.
        //
        fx_timer_set_abs(&timer, 5000, 20);
        ...;
}
```

fx_timer_set_rel

```
int fx_timer_set_rel(fx_timer_t* timer, uint32_t relative_start, uint32_t period);
```

Описание

Установка времени активации таймера. Время активации указывается в относительной форме (относительно момента вызова функции). Если таймер был активен, он неявно отменяется.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
timer	Указатель на таймер.
relative_start	Относительное время первого срабатывания таймера. Если это значение равно 0 – процедура активации таймера запускается немедленно (в зависимости от реализации допускается от 0 до 1 тика задержки).
period	Период последующих (2-го и далее) срабатываний таймера. Если это значение равно 0, таймер будет активирован один раз.

Возвращаемое значение

Возвращаемое значение	Описание
FX_TIMER_OK	Успешное завершение.
FX_TIMER _INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_TIMER_INVALID_OBJ	Некорректно проинициализированный объект таймера.
FX_TIMER_INVALID_TIMEOUT	Значение таймаута не находится в диапазоне 0-FX_TIMER_MAX_RELATIVE_TIMEOUT.
FX_TIMER_CONCURRENT_USE	Попытка одновременно установить один и тот же таймер с разных процессоров (неопределенное поведение). Таймер не был установлен.

Ремарки

В зависимости от реализации таймеров, данная функция может иметь задержку O(N), зависящую от количества таймеров в системе. В стандартных конфигурациях функция выполняется за фиксированное время.

Пример

```
fx_timer_t timer;

void my_thread(void* arg)
{
        //
        // Initialize timer to activate after 10 ticks first time and every 20
        // ticks after that.
        //
        fx_timer_set_rel(&timer, 10, 20);
        ...;
}
```

fx_timer_set_tick_count

```
uint32_t fx_timer_set_tick_count(uint32_t ticks);
```

Описание

Установка системного счетчика тиков. Из-за того, что эта функция влияет на время срабатывания всех установленных ранее таймеров, рекомендуется ее использование только в отладочных целях.

Контекст вызова

SPL = ANY

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	✅
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
ticks	Новое значение системного счетчика тиков.

Возвращаемое значение

Предыдущее значение системного счетчика тиков.

Ремарки

Данная функция является опциональной и может присутствовать не во всех реализациях модуля таймеров.

Пример

```
void my_thread(void* arg)
{
        fx_timer_set_tick_count(10000);
        ...;
}
```

Условные переменные

fx_cond_broadcast

```
int fx_cond_broadcast(fx_cond_t* cond);
```

Описание

Уведомление всех потоков, ожидающих условную переменную.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
cond	Указатель на условную переменную.

Возвращаемое значение

Возвращаемое значение	Описание
FX_COND_OK	Успешное завершение.
FX_COND_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_cond_init.
FX_COND_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_cond_t cond;
void my_thread(void* arg)
{
        //
        // Broadcasting notification.
        //
        fx_cond_broadcast(&cond);
}
```

fx_cond_deinit

```
int fx_cond_deinit(fx_cond_t* cond);
```

Описание

Удаление условной переменной. Все потоки, ожидающие условную переменную, возобновляют выполнение со статусом FX_THREAD_WAIT_DELETED. После вызова этой функции условная переменная не должна использоваться.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
cond	Указатель на условную переменную, которую надо удалить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_COND_OK	Успешное завершение.
FX_COND_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_cond_init.
FX_COND_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_cond_t cond;
void my_thread(void* arg)
{
        //
        // Delete the condition variable.
        //
        fx_cond_deinit(&cond);
}
```

fx_cond_init

```
int fx_cond_init(fx_cond_t* cond, fx_sync_policy_t policy);
```

Описание

Инициализация условной переменной.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	❌
Обработчик таймера	❌

Аргументы

Аргумент	Описание
cond	Указатель на условную переменную, которую надо проинициализировать.
policy	Политика уведомления ожидающих потоков. Определяется используемым слоем синхронизации. По умолчанию должно использоваться значение FX_SYNC_POLICY_DEFAULT. В случае явного указания политики могут поддерживаться следующие значения: 1. FX_SYNC_POLICY_FIFO – уведомление ожидающих потоков в том порядке, в котором они начали ожидание. 2. FX_SYNC_POLICY_PRIO – уведомление в первую очередь ожидателей с максимальным приоритетом. Так как поддержка различных политик зависит от конфигурации ОС, рекомендуется использовать значение по-умолчанию в тех случаях, когда требуется достичь максимальной переносимости между различными конфигурациями. Политика используется только для нешироковещательных уведомлений (когда необходимо уведомить не все потоки, ожидающие данного примитива).

Возвращаемое значение

Возвращаемое значение	Описание
FX_COND_OK	Успешное завершение.
FX_COND_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_COND_UNSUPPORTED_POLICY	Некорректный аргумент policy.

Ремарки

Нет.

Пример

```
fx_cond_t cond;
void my_thread(void* arg)
{
        //
        // Initialize the condition variable with FIFO policy.
        //
        fx_cond_init(&cond, FX_SYNC_POLICY_FIFO);
}
```

fx_cond_signal

```
int fx_cond_signal(fx_cond_t* cond);
```

Описание

Нотификация одного потока, ожидающего условную переменную. Если переменную ожидают несколько потоков, только один из них возобновит выполнение.

Контекст вызова

SPL <= SCHED_LEVEL

Контекст вызова	Ограничения
Инициализация	✅
Приложение	✅
Обработчик прерывания	❌
Отложенная процедура/DPC	✅
Обработчик таймера	✅

Аргументы

Аргумент	Описание
cond	Указатель на условную переменную.

Возвращаемое значение

Возвращаемое значение	Описание
FX_COND_OK	Успешное завершение.
FX_COND_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_cond_init.
FX_COND_INVALID_PTR	Некорректный (нулевой) указатель на объект.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_cond_t cond;
void my_thread(void* arg)
{
        //
        // Release one waiting thread.
        //
        fx_cond_signal(&cond);
}
```

fx_cond_timedwait

```
int fx_cond_timedwait(fx_cond_t* cond, fx_mutex_t* mutex, uint32_t tout);
```

Описание

Ожидание условной переменной с таймаутом. Мьютекс, переданный в качестве аргумента должен быть захвачен потоком перед вызовом функции (он атомарно освобождается после начала ожидания условной переменной). Мьютекс захватывается снова, после окончания ожидания (перед выходом из функции).

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
cond	Указатель на условную переменную.
mutex	Мьютекс для синхронизации доступа к условной переменной (должен быть захвачен потоком перед вызовом функции ожидания).
timeout	Таймаут ожидания в тиках таймера (обычно 1 тик = 1 или 10 мс). Для бесконечного таймаута следует оуказывать значение FX_THREAD_INFINITE_TIMEOUT. Таймаут не должен превышать значение FX_TIMER_MAX_RELATIVE_TIMEOUT.

Возвращаемое значение

Возвращаемое значение	Описание
FX_COND_OK	Успешное завершение.
FX_COND_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_cond_init.
FX_COND_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания.
FX_THREAD_WAIT_DELETED	Объект был удален во время ожидания.
FX_COND_INVALID_MUTEX	Нулевой указатель на мьютекс или вызывающая поток не является владельцем мьютекса.
FX_COND_MUTEX_ERROR	Отличный от успешного статус захвата мьютекса, при попытке его повторного захвата после окончания ожидания условной переменной.

FX_THREAD_INVALID_TIMEOUT	Значение таймаута не находится в диапазона 0-FX_THREAD_MAX_RELATIVE_TIMEOUT и он не равен FX_THREAD_INFINITE_TIMEOUT.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_cond_t cond;
fx_mutex_t mutex;

void my_thread(void* arg)
{
        //...
        fx_mutex_acquire(&mutex, NULL);
        fx_cond_timedwait(&cond, &mutex, 10);
        fx_mutex_release(&mutex);
}
```

fx_cond_wait

```
int fx_cond_wait(fx_cond_t* cond, fx_mutex_t* mutex, fx_event_t* cancel_event);
```

Описание

Ожидание условной переменной с отменяющим событием. Мьютекс, переданный в качестве аргумента должен быть захвачен потоком перед вызовом fx_cond_wait (он атомарно освобождается после начала ожидания условной переменной). Мьютекс захватывается снова, после окончания ожидания (перед выходом из функции).
Отменяющее событие может использоваться для прерывания ожидания.

Контекст вызова

SPL = LOW

Контекст вызова	Ограничения
Инициализация	✗
Приложение	✓
Обработчик прерывания	✗
Отложенная процедура/DPC	✗
Обработчик таймера	✗

Аргументы

Аргумент	Описание
cond	Указатель на условную переменную.
mutex	Мьютекс для синхронизации доступа к условной переменной (должен быть захвачен потоком перед вызовом функции ожидания).
cancel_event	Отменяющее событие. Может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
FX_COND_OK	Успешное завершение.
FX_COND_INVALID_OBJ	Объект не был корректно инициализирован с помощью функции fx_cond_init.
FX_COND_INVALID_PTR	Некорректный (нулевой) указатель на объект.
FX_THREAD_WAIT_CANCELLED	Ожидание было отменено из-за установки отменяющего события в активное состояние.
FX_THREAD_WAIT_DELETED	Один из объектов (условная переменная, или отменяющее событие, если оно было указано) был удален во время ожидания.
FX_COND_INVALID_MUTEX	Нулевой указатель на мьютекс или вызывающая поток не является владельцем мьютекса.
FX_COND_MUTEX_ERROR	Отличный от успешного статус захвата мьютекса, при попытке его повторного захвата после окончания ожидания условной переменной.

FX_EVENT_INVALID_OBJ	Отменяющее событие было указано (не равно NULL), но не является корректным объектом.

Ремарки

В результате выполнения функции вызывающий поток может быть вытеснен.

Пример

```
fx_cond_t cond;
fx_mutex_t mutex;
fx_event_t event;

void my_thread(void* arg)
{
        fx_mutex_acquire(&mutex, NULL);
        error = fx_cond_wait(&cond, &mutex, &event);
        fx_mutex_release(&mutex);
}
```

Таблица возвращаемых кодов ошибок

Таблица численных значений кодов ошибок для функций прикладного интерфейса FX-RTOS.

Символьное имя	Десятичное значение	16-ричное значение
FX_BARR_OK	00	0x00
FX_BARR_INVALID_PTR	16	0x10
FX_BARR_INVALID_OBJ	17	0x11
FX_BARR_ZERO_LIMIT	18	0x12
FX_BLOCK_POOL_OK	00	0x00
FX_BLOCK_POOL_INVALID_PTR	16	0x10
FX_BLOCK_POOL_INVALID_OBJ	17	0x11
FX_BLOCK_POOL_NO_MEM	18	0x12
FX_BLOCK_POOL_IMPROPER_ALIGN	19	0x13
FX_BLOCK_POOL_UNSUPPORTED_POLICY	20	0x14
FX_COND_OK	00	0x00
FX_COND_INVALID_PTR	16	0x10
FX_COND_INVALID_OBJ	17	0x11
FX_COND_UNSUPPORTED_POLICY	18	0x12
FX_COND_INVALID_MUTEX	19	0x13
FX_COND_MUTEX_ERROR	20	0x14
FX_COND_INVALID_TIMEOUT	21	0x15
FX_COND_INVALID_PARAMETER	22	0x16
FX_EVENT_OK	00	0x00
FX_EVENT_INVALID_PTR	01	0x01
FX_EVENT_INVALID_OBJ	02	0x02
FX_MEM_POOL_OK	00	0x00
FX_MEM_POOL_INVALID_PTR	01	0x01
FX_MEM_POOL_INVALID_OBJ	02	0x02
FX_MEM_POOL_ZERO_SZ	03	0x03
FX_MEM_POOL_NO_MEM	04	0x04
FX_MSGQ_OK	00	0x00
FX_MSGQ_INVALID_PTR	16	0x10
FX_MSGQ_INVALID_OBJ	17	0x11
FX_MSGQ_INVALID_BUF	18	0x12
FX_MSGQ_UNSUPPORTED_POLICY	19	0x13
FX_MUTEX_OK	00	0x00
FX_MUTEX_INVALID_PTR	16	0x10

FX_MUTEX_INVALID_OBJ	17	0x11
FX_MUTEX_UNSUPPORTED_POLICY	18	0x12
FX_MUTEX_INVALID_PRIORITY	19	0x13
FX_MUTEX_INVALID_TIMEOUT	20	0x14
FX_MUTEX_WRONG_OWNER	21	0x15
FX_MUTEX_RECURSIVE_LIMIT	22	0x16
FX_MUTEX_ABANDONED	23	0x17
FX_RWLOCK_OK	00	0x00
FX_RWLOCK_INVALID_PTR	16	0x10
FX_RWLOCK_INVALID_OBJ	17	0x11
FX_RWLOCK_UNSUPPORTED_POLICY	18	0x12
FX_RWLOCK_INVALID_TIMEOUT	19	0x13
FX_SEM_OK	00	0x00
FX_SEM_INVALID_PTR	16	0x10
FX_SEM_INVALID_OBJ	17	0x11
FX_SEM_UNSUPPORTED_POLICY	18	0x12
FX_SEM_INVALID_VALUE	19	0x13
FX_SEM_INVALID_TIMEOUT	20	0x14
FX_THREAD_OK	00	0x00
FX_THREAD_WAIT_CANCELLED	01	0x01
FX_THREAD_WAIT_DELETED	02	0x02
FX_THREAD_WAIT_INTERRUPTED	03	0x03
FX_THREAD_WAIT_TIMEOUT	04	0x04
FX_THREAD_INVALID_PTR	06	0x06
FX_THREAD_INVALID_ENTRY	07	0x07
FX_THREAD_INVALID_PRIO	08	0x08
FX_THREAD_INVALID_CPU	09	0x09
FX_THREAD_NO_STACK	10	0x0a
FX_THREAD_INVALID_OBJ	11	0x0b
FX_THREAD_INVALID_TIMEOUT	12	0x0c
FX_THREAD_INVALID_PARAM	13	0x0d
FX_THREAD_JOIN_SELF	14	0x0e
FX_THREAD_INVALID_TIMESLICE	15	0x0f
FX_TIMER_OK	00	0x00
FX_TIMER_ALREADY_CANCELLED	01	0x01
FX_TIMER_CONCURRENT_USE	02	0x02
FX_TIMER_INVALID_PTR	03	0x03
FX_TIMER_INVALID_OBJ	04	0x04

FX_TIMER_INVALID_TIMEOUT	05	0x05
FX_TIMER_INVALID_CALLBACK	06	0x06

Функции расширенного прикладного интерфейса

Расширенный интерфейс имеет два контекста применения, некоторые функции могут использоваться только в приложениях ядра (kernel applications), эти функции отличаются префиксом ex_.

Функции с префиксом fx_ могут использоваться только в приложениях, в непривилегированном режиме процессора.

Асинхронная передача сообщений (опциональный компонент)

ex_named_mq_init

void ex_named_mq_init(ex_obj_pool_t pool, unsigned* **prio);**

Описание

Установка параметров для компонента, реализующего асинхронную передачу сообщений.

Аргументы

Аргумент	Описание
pool	Указатель на предварительно инициализированный пул ресурсов, из которого будет происходить выделение памяти для объектов очереди, а также для сообщений.
prio	Потолок приоритета для внутренних мьютексов, защищающих базу данных имён объектов. Всякий раз при поиске объекта по имени приоритет вызывающего потока будет повышаться до указанного уровня. Если потолок приоритета не используется, следует использовать значение FX_MUTEX_CEILING_DISABLED.

Возвращаемое значение

Нет.

Ремарки

При выборе значения потолка приоритета следует учитывать максимальный приоритет потоков для процесса.

fx_mq_close

int fx_mq_close(fx_mq_t mq);*

Описание

Закрытие ссылки на открытую очередь сообщений.

Аргументы

Аргумент	Описание
mq	Открытая очередь сообщений.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение.
FX_MSGQ_INVALID_OBJ	Некорректный идентификатор объекта.

Ремарки

Нет.

fx_mq_open

int fx_mq_open(
fx_mq_t mq, char* name,*
unsigned max_msgs, unsigned max_msg_sz, unsigned flags);

Описание

Открытие именованного объекта "очередь сообщений" для посылки или приёма из нее асинхронных сообщений.

Аргументы

Аргумент	Описание
mq	Указатель на объект "очередь", который должен быть открыт.
name	Имя очереди.
max_msgs	Максимальное количество сообщений в очереди. Это число не превышает значение EX_MQ_MSG_MAX задаваемое в конфигурационных опциях ядра (если указано значение больше, будет использоваться EX_MQ_MSG_MAX).
max_msg_sz	Максимальный размер сообщения в очереди.
flags	Опции открытия объекта или 0. Допустимы следующие опции: • FX_MQ_CREATE - создать новый порт с таким именем если таковой не существует на момент вызова функции. Открыть существующий порт, если он существует. • FX_MQ_EXCLUSIVE - Завершить вызов с ошибкой если порт уже существует. Этот флаг подразумевает также FX_MQ_CREATE.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение.
FX_MODULE_NO_RESOURCES	Недостаточно ресурсов для создания нового объекта.
FX_MSGQ_INVALID_OBJ	Порт с указанным именем не существует и не был указан флаг FX_MQ_CREATE. Порт с указанным именем существует, но был указан флаг FX_MQ_EXCLUSIVE.
FX_MSGQ_INVALID_BUF	Некорректный буфер имени: недоступный процессу или содержащий некорректное имя.

Ремарки

Во время работы этой функции приоритет вызывающего потока может быть повышен до указанного в функции ex_named_mq_init.

fx_mq_timedreceive

int fx_mq_timedreceive(fx_mq_t mq, void* buf, size_t sz, uint32_t timeout);*

Описание

Прием сообщения из очереди в локальный буфер. Если в очереди есть сообщения (буферизованные) сообщение будет скопировано и возврат из функции произойдет немедленно. В случае, если очередь пуста, вызывающий поток будет заблокирован до тех пор, пока какой-либо другой поток не отправит сообщение в очередь.

Аргументы

Аргумент	Описание
mq	Открытый объект очереди сообщений.
buf	Буфер, в который будет помещено принятое сообщение. ОС не накладывает никаких ограничений на содержимое сообщений, их формат определяется взаимным соглашением клиента и сервера.
sz	Размер сообщения. Это значение не должно превышать указанное при открытии очереди.
timeout	Таймаут ожидания.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение.
FX_MSGQ_INVALID_BUF	Некорректный буфер приёма или указанный размер сообщения превышает максимальный для данного объекта очереди
FX_MSGQ_INVALID_OBJ	Некорректный идентификатор объекта.
FX_THREAD_WAIT_TIMEOUT	В очереди не появилось сообщений за указанный период времени.
FX_THREAD_WAIT_INTERRUPTED	Ожидание было прервано сигналом.

Ремарки

Нет.

fx_mq_timedsend

int fx_mq_timedsend(fx_mq_t mq, void* buf, size_t sz, uint32_t timeout);*

Описание

Отправка сообщения из локального буфера в очередь. Если в очереди есть место для сообщения оно будет скопировано и возврат из функции произойдет немедленно. В случае, если очередь заполнена, вызывающий поток будет заблокирован до тех пор, пока в очереди не появится место.

Аргументы

Аргумент	Описание
mq	Открытый объект очереди сообщений.
buf	Буфер сообщения. ОС не накладывает никаких ограничений на содержимое сообщений, их формат определяется взаимным соглашением клиента и сервера.
sz	Размер сообщения. Это значение не должно превышать указанное при открытии очереди.
timeout	Таймаут ожидания.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение.
FX_MSGQ_INVALID_BUF	Некорректный буфер сообщения или указанный размер сообщения превышает максимальный для данного объекта очереди.
FX_MSGQ_INVALID_OBJ	Некорректный идентификатор объекта.
FX_THREAD_WAIT_TIMEOUT	В заполненной очереди не появилось место за указанный период времени.
FX_THREAD_WAIT_INTERRUPTED	Ожидание было прервано сигналом.

Ремарки

Нет.

fx_mq_unlink

int fx_mq_unlink(char name);*

Описание

Удаление очереди сообщений с указанным именем. После удаления становится невозможно открытие очереди по имени, однако имеющиеся открытые ссылки на данную очередь продолжат работать. Удаление очереди произойдет после закрытия последний ссылки на нее.

Аргументы

Аргумент	Описание
name	Имя удаляемой очереди.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSGQ_OK	Успешное завершение.
FX_MSGQ_INVALID_BUF	Некорректный буфер имени: недоступный процессу или содержащий некорректное имя.
FX_MSGQ_INVALID_OBJ	Порт с данным именем не существует.

Ремарки

Во время работы этой функции приоритет вызывающгго потока может быть повышен до указанного в функции ex_named_mq_init.

Именованные семафоры

ex_named_sems_init

void ex_named_sems_init(ex_obj_pool_t pool, unsigned prio);*

Описание

Установка параметров для компонента, реализующего именованные семафоры.

Аргументы

Аргумент	Описание
pool	Указатель на предварительно инициализированный пул ресурсов, из которого будет происходить выделение памяти для объектов "именованный семафор".
prio	Потолок приоритета для внутренних мьютексов, защищающих базу данных имён объектов. Всякий раз при поиске объекта по имени приоритет вызывающего потока будет повышаться до указанного уровня. Если потолок приоритета не используется, следует использовать значение FX_MUTEX_CEILING_DISABLED.

Возвращаемое значение

Нет.

Ремарки

При выборе значения потолка приоритета следует учитывать максимальный приоритет потоков для данного процесса.

fx_sem_close

int fx_sem_close(fx_sem_t sem);*

Описание

Закрытие ссылки на семафор.

Аргументы

Аргумент	Описание
sem	Открытый именованный семафор.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM_INVALID_OBJ	Некорректный идентификатор объекта.

Ремарки

Нет.

fx_sem_open

int fx_sem_open(
fx_sem_t sem, char* name,*
unsigned init, unsigned max_val, unsigned flags);

Описание

Открытие именованного семафора. После открытия и получения идентификатора, для именованного семафора могут вызываться те же функции API, что и для локальных семафоров.

Аргументы

Аргумент	Описание
sem	Указатель на объект семафор, который должен быть открыт.
name	Имя семафора.
init	Начальное значение семафора
max_val	Максимальное значение счетчика семафора.
flags	Опции открытия объекта или 0. Допустимы следующие опции: • FX_SEM_CREATE - создать новый порт с таким именем если таковой не существует на момент вызова функции. Открыть существующий порт, если он существует. • FX_SEM_EXCLUSIVE - Завершить вызов с ошибкой если порт уже существует. Этот флаг подразумевает также FX_SEM_CREATE.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_PROCESS_NO_RESOURCES	Недостаточно ресурсов для создания нового объекта.
FX_SEM_INVALID_OBJ	Семафор с указанным именем не существует и не был указан флаг FX_SEM_CREATE. Семафор с указанным именем существует, но был указан флаг FX_SEM_EXCLUSIVE.
FX_SEM_INVALID_PTR	Некорректный буфер имени: недоступный процессу или содержащий некорректное имя.

Ремарки

Во время работы этой функции приоритет вызывающего потока может быть повышен до указанного в функции ex_named_sems_init.

fx_sem_unlink

int fx_sem_unlink(char name);*

Описание

Удаление семафора с указанным именем. После удаления становится невозможно открытие семафора по имени, однако имеющиеся открытые ссылки на данный объект продолжат работать. Удаление произойдет после закрытия последний ссылки на семафор.

Аргументы

Аргумент	Описание
name	Имя удаляемого семафора.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SEM_OK	Успешное завершение.
FX_SEM_INVALID_PTR	Некорректный буфер имени: недоступный процессу или содержащий некорректное имя.
FX_SEM_INVALID_OBJ	Некорректный идентификатор объекта.

Ремарки

Во время работы этой функции приоритет вызывающего потока может быть повышен до указанного в функции ex_named_sems_init.

Прерывания

fx_interrupt_attach

int fx_interrupt_attach(fx_interrupt_t interrupt);*

Описание

Присоединение инициализированного объекта "прерывание" к примитиву синхронизации, который был указан при инициализации.

Аргументы

Аргумент	Описание
interrupt	Объект "прерывание", который необходимо присоединить.

Возвращаемое значение

Возвращаемое значение	Описание
FX_INTERRUPT_OK	Успешное завершение.
FX_INTERRUPT_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_ISR_ALREADY_ATTACHED	Данный вектор прерывания уже был присоединен к примитиву синхронизации.

Ремарки

После успешного завершения данной функции объект синхронизации может быть установлен в активное состояние обработчиком прерывания. После возникновения прерывания данный вектор маскируется, до тех пор пока пользователь не размаскирует источник с помощью функции fx_interrupt_unmask.

fx_interrupt_detach

int fx_interrupt_detach(fx_interrupt_t interrupt);*

Описание

Отсоединение объекта синхронизации (указанного при инициализации) от источника прерываний.

Аргументы

Аргумент	Описание
interrupt	Отсоединяемый объект "прерывание".

Возвращаемое значение

Возвращаемое значение	Описание
FX_INTERRUPT_OK	Успешное завершение.
FX_INTERRUPT_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_INTERRUPT_INVALID_OBJ	Данный объект "прерывание" не был присоединен к какому-либо вектору.

Ремарки

Данная функция гарантирует, что все процессы относящиеся к объекту "прерывание" с данным вектором завершены, используемые ресурсы (напр. семафор) могут быть освобождены.

fx_interrupt_init

int fx_interrupt_init(
fx_interrupt_t interrupt, fx_sem_t* sem, unsigned irq);*

Описание

Инициализация объекта "прерывание", который связывает источник прерывания и объект синхронизации (семафор).

Аргументы

Аргумент	Описание
interrupt	Объект "прерывание", который необходимо инициализировать.
sem	Указатель на предварительно инициализированный объект "семафор".
irq	Вектор прерывания, к которому следует присоединить семафор.

Возвращаемое значение

Возвращаемое значение	Описание
FX_INTERRUPT_OK	Успешное завершение.
FX_INTERRUPT_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_ISR_INVALID_VECTOR	Неверный (неподдерживаемый) вектор прерывания.
FX_INTERRUPT_INVALID_OBJ	Недопустимый объект синхронизации (семафор).

Ремарки

Нет.

fx_interrupt_mask

int fx_interrupt_mask(fx_interrupt_t interrupt, bool mask);*

Описание

Установка маски источника прерывания.

Аргументы

Аргумент	Описание
interrupt	Объект "прерывание", для вектора которого нужно установить маску.
Mask	Маска прерывания. True для маскировки вектора, false для размаскировки.

Возвращаемое значение

Возвращаемое значение	Описание
FX_INTERRUPT_OK	Успешное завершение.
FX_INTERRUPT_INVALID_PTR	Недопустимый (нулевой) указатель на объект.
FX_INTERRUPT_INVALID_OBJ	Данный объект "прерывание" не был присоединен к какому-либо вектору.

Ремарки

Нет.

Процессы

ex_process_init

```
int ex_process_init(ex_ process_t* process, ex_obj_pool_t* pool, void (*func)(ex_process_t*));
```

Описание

Инициализация объекта, соответствующего непривилегированному процессу.

Аргументы

Аргумент	Описание
process	Указатель на выделенный пользователем объект, соответствующий процессу.
pool	Инициализированный пул объектов, из которого будет происходить выделение памяти (см.ex_obj_pool_init). Один пул может использоваться как источник ресурсов для нескольких процессов, в этом случае они будут разделять между собой ресурсы пула.
func	Пользовательская функция, которая будет вызвана при завершении всех потоков процесса. Опциональный аргумент, может быть NULL.

Возвращаемое значение

Возвращаемое значение	Описание
EX_PROCESS_OK	Успешное завершение.
EX_ PROCESS_INVALID	Некорректный (нулевой) указатель на объект.
EX_PROCESS_NO_MEM	Недостаточно памяти для внутренних структур данных процесса.

Ремарки

Для каждого процесса инициализация должна быть выполнена только один раз, при последующих перезапусках процесса, которые могут произойти в случае аварийного завершения работы последнего, заново инициализировать объект не требуется.
Данная функция также выделяет необходимые структуры для защиты памяти, связанные с процессом (такие как таблицы страниц в случае использования MMU), поэтому ее выполнение может иметь побочные эффекты.
Перед использованием данной функции память, используемая объектом, должна быть обнулена.

ex_process_kill

int ex_process_kill(ex_process_t process);*

Описание

Принудительное завершение всех потоков, выполняющихся в адресном пространстве процесса и освобождение всех занимаемых им ресурсов.

Аргументы

Аргумент	Описание
process	Указатель на объект, соответствующий процессу.

Возвращаемое значение

Возвращаемое значение	Описание
EX_PROCESS_OK	Успешное завершение.
EX_PROCESS_INVALID	Некорректный (нулевой) указатель на объект.

Ремарки

Функция выполняется асинхронно.

ex_process_self

ex_process_t ex_process_self(void);*

Описание

Получение указателя на текущий выполняемый процесс.

Аргументы

Нет.

Возвращаемое значение

Указатель на текущий выполняемый процесс.

Ремарки

Функция может вызываться только в режиме ядра в контексте выполняемого процесса. Например, в обработчиках PFC для реализации поведения, зависящего от вызывающего процесса.

ex_process_set_ priority_quota

void ex_process_set_priority_quota(ex_process_t process, unsigned prio);*

Описание

Указание максимального приоритета для потоков, выполняющихся в адресном пространстве процесса.

Аргументы

Аргумент	Описание
process	Указатель на выделенный пользователем объект, соответствующий процессу.
prio	Максимальный приоритет потоков.

Возвращаемое значение

Нет.

Ремарки

Помимо невозможности создания потоков с приоритетом, превышающим указанный, ядром ОС отклоняется также возможность создания мьютексов с высоким потолком приоритета, позволяющим обойти ограничение.

ex_process_start

int ex_process_start(
ex_process_t process, void (**entry)(void**), void* arg, unsigned prio,*
void stk, size_t stk_sz);*

Описание

Запуск процесса. Функция создает первый поток, который будет выполняться в непривилегированном режиме в адресном пространстве соответствующем указанному процессу.

Аргументы

Аргумент	Описание
process	Указатель на выделенный пользователем и предварительно инициализированный объект, соответствующий процессу.
entry	Точка входа для вновь создаваемого потока. Адрес должен находиться в области памяти, доступной процессу, в противном случае попытка запуска потока сразу же приведет к его аварийному завершению.
arg	Аргумент создаваемого потока.
prio	Приоритет создаваемого потока. Этот приоритет не должен быть выше (численно меньше), чем максимальный приоритет потоков, создание которых разрешено для данного процесса. (см.ex_process_set_ priority_quota).
stk	Базовый адрес стека. Адрес должен находиться в области памяти, доступной процессу, в противном случае попытка запуска процесса сразу же приведет к его аварийному завершению.
stk_sz	Размер стека для вновь создаваемого потока.

Возвращаемое значение

Возвращаемое значение	Описание
EX_PROCESS_OK	Успешное завершение.
EX_PROCESS_INVALID	Некорректный (нулевой) указатель на объект.
EX_PROCESS_1ST_THREAD_ERROR	Ошибка создания потока (некорректное значение потоковой функции, приоритета или стека).

Ремарки

В случае аварийного завершения работы процесса, данная функция может использоваться для его перезапуска. Рестарт процесса происходит в том же адресном пространстве с теми же ресурсами и ограничениями, с которыми он был инициализирован.

ex_process_virtual_alloc

bool ex_process_virtual_alloc(
ex_process_t process, void* vaddr, void* paddr, size_t sz, bool ro);*

Описание

Установка прав доступа к региону виртуальных адресов в адресном пространстве процесса.

Аргументы

Аргумент	Описание
process	Указатель на выделенный пользователем объект, соответствующий процессу.
vaddr	Базовый виртуальный адрес региона (адрес, который должен быть доступен для потоков, выполняющихся в адресном пространстве процесса).
paddr	Физический адрес, который будет соответствовать указанному виртуальному адресу.
sz	Размер региона. Следует учитывать, что указанный размер округляется до размера аппаратной страницы, поэтому доступный процессу регион может оказаться больше, чем указано.
ro	Установка прав доступа "только для чтения". Если данный аргумент равен false, тогда регион доступен как для чтения, так и для записи.

Возвращаемое значение

True в случае успешного завершения, false в противном случае.

Ремарки

Для систем, которые используют MPU (memory protection unit) и не поддерживают трансляцию адресов, виртуальный и физический адрес должны совпадать. Также в случае использования MPU размер региона округляется до ближайшей степени числа 2, начиная с минимального размера, поддерживаемого аппаратно.

Прочие функции

ex_obj_pool_init

void ex_obj_pool_init(ex_obj_pool_t op, void* pool, size_t pool_sz);*

Описание

Инициализация пула для динамического выделения памяти.

Аргументы

Аргумент	Описание
op	Объект пула для инициализации.
pool	Указатель на память, которой управляет пул. Обычно эта память выделяется статически.
pool_sz	Размер блока памяти.

Возвращаемое значение

Нет.

Ремарки

Нет.

Сигналы

fx_signal_get_thread_id

unsigned fx_signal_get_thread_id(void);

Описание

Получение уникального идентификатора вызывающего потока. Полученный идентификатор идентифицирует данный поток в системе, а не в процессе, поэтому может использоваться для межпроцессного взаимодействия.

Аргументы

Нет.

Возвращаемое значение

Системный идентификатор текущего потока.

Ремарки

Нет.

fx_signal_mask

uint32_t fx_signal_mask(uint32_t sig_set, bool mask);

Описание

Установка маски для сигнала с заданным идентификатором.

Аргументы

Аргумент	Описание
sig_set	Битовая маска сигналов для которых задается маска.
mask	Новое значение маски для указанного набора сигналов (true для маскировки и false для размаскировки).

Возвращаемое значение

Предыдущее состояние маски сигналов.

Ремарки

Нет.

fx_signal_post

int fx_signal_post(fx_thread_t target, unsigned sig_no);*

Описание

Отправка сигнала указанному потоку.

Аргументы

Аргумент	Описание
target	Идентификатор потока в текущем процессе, которому будет направлен сигнал.
sig_no	Идентификатор сигнала. Допустимы следующие значения: • FX_SIGNAL_ILL - недопустимая операция • FX_SIGNAL_SEGV - ошибка доступа к памяти • FX_SIGNAL_TRAP - отладочное исключение • FX_SIGNAL_FPE - ошибка FPU • FX_SIGNAL_USR1 - пользовательский сигнал 1 • FX_SIGNAL_USR2 - пользовательский сигнал 2 Пользовательские сигналы могут использоваться приложением для внутренних нужд, эти сигналы не используются ОС. Остальные сигналы генерируются ядром в ответ на различные ошибки.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SIGNAL_OK	Успешное завершение.
FX_SIGNAL_INVALID_OBJ	Некорректный идентификатор потока.
FX_SIGNAL_UNKNOWN_ID	Некорректный идентификатор сигнала.

Ремарки

Кроме определенных для пользователя сигналов с идентификаторами FX_SIGNAL_USR1/USR2 потоку могут быть также направлены системные сигналы.

fx_signal_post_by_id

int fx_signal_post_by_id(unsigned thread_id, unsigned sig_no);

Описание

Отправка сигнала потоку по его системному идентификатору. Может использоваться как для отправки сигналов потокам в текущем процессе, так и для межпроцессного взаимодействия.

Аргументы

Аргумент	Описание
thread_id	Системный идентификатор потока, которому следует направить сигнал.
sig_no	Идентификатор сигнала. Допустимы следующие значения: • FX_SIGNAL_ILL - недопустимая операция • FX_SIGNAL_SEGV - ошибка доступа к памяти • FX_SIGNAL_TRAP - отладочное исключение • FX_SIGNAL_FPE - ошибка FPU • FX_SIGNAL_USR1 - пользовательский сигнал 1 • FX_SIGNAL_USR2 - пользовательский сигнал 2 Пользовательские сигналы могут использоваться приложением для внутренних нужд, эти сигналы не используются ОС. Остальные сигналы генерируются ядром в ответ на различные ошибки.

Возвращаемое значение

Возвращаемое значение	Описание
FX_SIGNAL_OK	Успешное завершение.
FX_SIGNAL_INVALID_OBJ	Некорректный идентификатор потока.
FX_SIGNAL_UNKNOWN_ID	Некорректный идентификатор сигнала.

Ремарки

Нет.

fx_signal_return

void fx_signal_return(unsigned sig_no, void context);*

Описание

Возврат из обработчика сигнала в прерванный поток.

Аргументы

Аргумент	Описание
sig_no	Идентификатор сигнала из которого производится возврат. Этот сигнал будет размаскирован одновременно с восстановлением контекста.
context	Контекст процессора который следует восстановить. Этот параметр приходит в качестве параметра в обработчик сигнала.

Возвращаемое значение

Нет.

Ремарки

Если были указаны неверные параметры, поток может быть принудительно завершен.

fx_signal_setup

```
int fx_signal_setup(unsigned sig_no, void (*func)(unsigned, void*));
```

Описание

Установка обработчика сигнала.

Аргументы

Аргумент	Описание
sig_no	Идентификатор сигнала. Допустимы следующие значения: • FX_SIGNAL_ILL - недопустимая операция • FX_SIGNAL_SEGV - ошибка доступа к памяти • FX_SIGNAL_TRAP - отладочное исключение • FX_SIGNAL_FPE - ошибка FPU • FX_SIGNAL_USR1 - пользовательский сигнал 1 • FX_SIGNAL_USR2 - пользовательский сигнал 2 Пользовательские сигналы могут использоваться приложением для внутренних нужд, эти сигналы не используются ОС. Остальные сигналы генерируются ядром в ответ на различные ошибки.
func	Функция обработчик сигнала, которая вызывается синхронно в контексте потока, которому был направлен сигнал. В качестве первого аргумента она принимает номер сигнала, в качестве второго - указатель на сохраненный в стеке контекст, который был в момент возникновения ошибки, что позволяет обработчику проанализировать причины сбоя. Допустимы также два специальных значения: • FX_SIGNAL_DFL - установить обработчик по-умолчанию • FX_SIGNAL_IGN - игнорировать сигнал

Возвращаемое значение

Возвращаемое значение	Описание
FX_SIGNAL_OK	Успешное завершение.
FX_SIGNAL_UNKNOWN_ID	Неизвестный номер сигнала.

Ремарки

Нет.

fx_signal_suspend

int fx_signal_suspend(uint32_t sig_set, uint32_t timeout);

Описание

Приостановка вызывающего потока до тех пор, пока им не будут получены все указанные в маске сигналы. Все ожидаемые сигналы должны быть предварительно замаскированы.

Аргументы

Аргумент	Описание
sig_set	Битовая маска сигналов для ожидания.
timout	Таймаут ожидания.

Возвращаемое значение

Возвращаемое значение	Описание
FX_THREAD_OK	Успешное завершение.
FX_SIGNAL_INVALID_OBJ	Указанные в битовой маске сигналы не замаскированы.
FX_THREAD_WAIT_TIMEOUT	Истек таймаут ожидания.
FX_THREAD_WAIT_INTERRUPTED	Ожидание было прервано другим незамаскированным сигналом, не входящим в ожидаемую группу.

Ремарки

Нет.

Синхронная передача сообщений

ex_named_msg_ports_init

void ex_named_msg_ports_init(ex_obj_pool_t pool, unsigned prio);*

Описание

Установка параметров для компонента, реализующего синхронную передачу сообщений.

Аргументы

Аргумент	Описание
pool	Указатель на предварительно инициализированный пул ресурсов, из которого будет происходить выделение памяти для объектов "порт сообщений".
prio	Потолок приоритета для внутренних мьютексов, защищающих базу данных имён объектов. Всякий раз при поиске объекта по имени приоритет вызывающего потока будет повышаться до указанного уровня. Если потолок приоритета не используется, следует использовать значение FX_MUTEX_CEILING_DISABLED.

Возвращаемое значение

Нет.

Ремарки

При выборе значения потолка приоритета следует учитывать максимальный приоритет для процесса.

fx_msg_port_close

int fx_msg_port_close(fx_msg_port_t port);*

Описание

Закрытие ссылки на открытый порт синхронных сообщений.

Аргументы

Аргумент	Описание
port	Открытый порт сообщений.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_MSG_PORT_INVALID_OBJ	Некорректный идентификатор порта.

Ремарки

Нет.

fx_msg_port_open

int fx_msg_port_open(fx_msg_port_t port, char* name, unsigned flags);*

Описание

Открытие именованного объекта "порт сообщений" для посылки или приёма из него синхронных сообщений.

Аргументы

Аргумент	Описание
port	Указатель на объект "порт", который должен быть открыт.
name	Имя порта.
flags	Опции открытия объекта или 0. Допустимы следующие опции: • FX_MSG_PORT_CREATE - создать новый порт с таким именем если таковой не существует на момент вызова функции. Открыть существующий порт, если он существует. • FX_MSG_PORT_EXCLUSIVE - Завершить вызов с ошибкой если порт уже существует. Этот флаг подразумевает также FX_MSG_PORT_CREATE. • FX_MSG_PORT_HEADER - Включить в сообщение определяемый ядром заголовок.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_MODULE_NO_RESOURCES	Недостаточно ресурсов для создания нового объекта.
FX_MSG_PORT_INVALID_OBJ	Порт с указанным именем не существует и не был указан флаг FX_MSG_PORT_CREATE. Порт с указанным именем существует, но был указан флаг FX_MSG_PORT_EXCLUSIVE.
FX_MSG_PORT_INVALID_BUF	Некорректный буфер имени: недоступный процессу или содержащий некорректное имя.

Ремарки

Во время работы этой функции приоритет вызывающего потока может быть повышен до указанного в функции ex_named_msg_ports_init.

fx_msg_port_unlink

int fx_msg_port_unlink(char name);*

Описание

Удаление порта сообщений с указанным именем. После удаления становится невозможно открытие порта по имени, однако имеющиеся открытые ссылки на данный порт продолжат работать. Удаление порта произойдет после закрытия последний ссылки на него.

Аргументы

Аргумент	Описание
name	Имя удаляемого порта.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_MSG_PORT_INVALID_BUF	Некорректный буфер имени: недоступный процессу или содержащий некорректное имя.
FX_MSG_PORT_INVALID_OBJ	Порт с данным именем не существует.

Ремарки

Во время работы этой функции приоритет вызывающего потока может быть повышен до указанного в функции ex_named_msg_ports_init.

fx_msg_receive

int fx_msg_receive(
fx_msg_port_t port, void* data, size_t buf_sz,*
size_t max_reply_sz, size_t* req_sz, bool block);*

Описание

Получение сообщения из порта. Вызывающий поток блокируется тех пор, пока не будет получено сообщение.

Аргументы

Аргумент	Описание
port	Указатель на объект "порт", который должен быть предварительно открыт. Порт должен быть открыт единожды любым потоком, выполняющимся в контексте данного процесса, после этого все потоки могут принимать сообщения.
data	Буфер для приёма сообщения. ОС не накладывает никаких ограничений на содержимое сообщений, их формат определяется взаимным соглашением клиента и сервера. Один и тот же буфер используется как для запроса, так и для ответа.
buf_sz	Размер буфера (максимальный размер сообщения, которое может принять данный поток).
max_reply_sz	Указатель на переменную, где будет сохранен максимальный размер ответа, который может принять клиент (размер буфера вызывающего потока).
req_sz	Указатель на переменную, где будет сохранен фактический размер запроса.
block	Если это значение равно false, функция немедленно возвращает управление, если порт не содержит ожидающего клиента в момент вызова (неблокирующий запрос).

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_THREAD_INVALID_OBJ	Некорректный порт.
FX_MSG_PORT_INVALID_BUF	Некорректный указатель на буфер или переменные для получения размеров запроса и ответа.
FX_MSG_PORT_TX_ABORTED	Клиент был найден, но произошла ошибка транзакции: • размер буфера недостаточен для приема запроса • клиент разблокировался и отменил запрос во время копирования
FX_THREAD_WAIT_TIMEOUT	Не найден клиент при неблокирующем вызове.
FX_THREAD_WAIT_INTERRUPTED	Выполнение функции было прервано сигналом.

Ремарки

Поток может обрабатывать только одно сообщение в каждый момент времени, перед повторным вызовом данной функции следует вызвать вызвать fx_msg_reply.
Если порт был создан с флагом FX_MSG_PORT_HEADER, тогда пересылаемое сообщение должно иметь зарезервированное место для заголовка. Заголовок имеет тип ker_msg_header_t и зависит от конфигурации ядра.

fx_msg_receive_buf

int fx_msg_receive_buf(
fx_msg_port_t port, fx_msg_buffer_t* vect, size_t vect_sz, bool block);*

Описание

Получение сообщения состоящего из нескольких буферов из порта. Вызывающий поток блокируется тех пор, пока не будет получено сообщение.

Аргументы

Аргумент	Описание
port	Указатель на объект "порт", который должен быть предварительно открыт. Порт должен быть открыт единожды любым потоком, выполняющимся в контексте данного процесса, после этого все потоки могут принимать сообщения.
vect	Массив структур имеющих тип fx_msg_buffer_t и описывающих буфер сообщения. Тип fx_msg_buffer_t определен как: **typedef struct _fx_msg_buffer_t** **{** **void** buf;* **size_t buf_sz;** **size_t req_sz;** **size_t reply_sz;** **}** **fx_msg_buffer_t;.** buf - Указатель на буфер. buf_sz - Размер буфера (максимальный размер ответа, который может принять данный поток). req_sz - Размер запроса. Если размер запроса меньше, чем размер ожидаемого ответа, это значение меньше, чем buf_sz. reply_sz - Фактический размер ответа сервера. Это число не превышает размера буфера.
vect_sz	Количество буферов в массиве vect. В текущей версии ядра поддерживается отправка только одного или двух буферов.
block	Если это значение равно false, функция немедленно возвращает управление, если порт не содержит ожидающего клиента в момент вызова (неблокирующий запрос).

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_THREAD_INVALID_OBJ	Некорректный порт.
FX_MSG_PORT_INVALID_BUF	Некорректный указатель на буфер или переменные для получения размеров запроса и ответа.
FX_MSG_PORT_TX_ABORTED	Клиент был найден, но произошла ошибка транзакции: • размер буфера недостаточен для приема запроса • клиент разблокировался и отменил запрос во время копирования
FX_THREAD_WAIT_TIMEOUT	Не найден клиент при неблокирующем вызове.
FX_THREAD_WAIT_INTERRUPTED	Выполнение функции было прервано сигналом.

Ремарки

Поток может обрабатывать только одно сообщение в каждый момент времени, перед повторным вызовом данной функции следует вызвать вызвать fx_msg_reply_buf.
Если порт был создан с флагом FX_MSG_PORT_HEADER, тогда пересылаемое сообщение должно иметь зарезервированное место для заголовка. Заголовок имеет тип ker_msg_header_t и зависит от конфигурации ядра. Заголовок пересылается как часть буфера 0.

fx_msg_reply

int fx_msg_reply(void data, size_t buf_sz);*

Описание

Ответ на полученное сообщение. После ответа поток, вызвавший fx_msg_send разблокируется и продолжит выполнение. Эта функция должна быть вызвана в контексте того потока, который получил сообщение с помощью fx_msg_receive.

Аргументы

Аргумент	Описание
data	Буфер содержащий сообщение. ОС не накладывает никаких ограничений на содержимое сообщений, их формат определяется взаимным соглашением клиента и сервера. Один и тот же буфер используется как для запроса, так и для ответа.
buf_sz	Размер буфера.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_MSG_PORT_NO_CLIENT	Нет сообщений, полученных данным потоком, либо клиент отменил запрос между получением сообщения и ответом на него.
FX_MSG_PORT_INVALID_BUF	Некорректный буфер. Если есть ожидающий клиент, можно попытаться вызвать функцию повторно с другим буфером, запрос клиента не отменяется.

Ремарки

Поток может обрабатывать только одно сообщение в каждый момент времени.

fx_msg_reply_buf

int fx_msg_reply_buf(void fx_msg_buffer_t vect, size_t* **vect_sz);**

Описание

Ответ на полученное сообщение. После ответа поток, вызвавший fx_msg_send_buf разблокируется и продолжит выполнение. Эта функция должна быть вызвана в контексте того потока, который получил сообщение с помощью fx_msg_receive_buf.

Аргументы

Аргумент	Описание
vect	Массив структур имеющих тип fx_msg_buffer_t и описывающих буфер сообщения. Тип fx_msg_buffer_t определен как: **typedef struct _fx_msg_buffer_t** **{** **void** buf;* **size_t buf_sz;** **size_t req_sz;** **size_t reply_sz;** **}** **fx_msg_buffer_t;.** buf - Указатель на буфер. buf_sz - Размер буфера (максимальный размер ответа, который может принять данный поток). req_sz - Размер запроса. Если размер запроса меньше, чем размер ожидаемого ответа, это значение меньше, чем buf_sz. reply_sz - Фактический размер ответа сервера. Это число не превышает размера буфера.
vect_sz	Количество буферов в массиве vect. В текущей версии ядра поддерживается отправка только одного или двух буферов.

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_MSG_PORT_NO_CLIENT	Нет сообщений, полученных данным потоком, либо клиент отменил запрос между получением сообщения и ответом на него.
FX_MSG_PORT_INVALID_BUF	Некорректный буфер. Если есть ожидающий клиент, можно попытаться вызвать функцию повторно с другим буфером, запрос клиента не отменяется.

Ремарки

Поток может обрабатывать только одно сообщение в каждый момент времени.

fx_msg_send

int fx_msg_send(
fx_msg_port_t port, void* data, size_t buf_sz, size_t req_sz,*
size_t reply_sz, bool block);*

Описание

Отправка сообщения в порт. Вызывающий поток блокируется тех пор, пока другой поток не получит сообщение из этого порта не ответит на него.

Аргументы

Аргумент	Описание
port	Указатель на объект "порт", который должен быть предварительно открыт. Порт должен быть открыт единожды любым потоком, выполняющейся в контексте данного процесса, после этого все потоки могут посылать в порт сообщения.
data	Буфер содержащий сообщение. ОС не накладывает никаких ограничений на содержимое сообщений, их формат определяется взаимным соглашением клиента и сервера. Один и тот же буфер используется как для запроса, так и для ответа.
buf_sz	Размер буфера (максимальный размер ответа, который может принять данный поток).
req_sz	Размер запроса. Если размер запроса меньше, чем размер ожидаемого ответа, это значение меньше, чем buf_sz.
reply_sz	Указатель на переменную, где будет сохранен фактический размер ответа сервера. Это число не превышает размера буфера.
block	Если это значение равно false, функция немедленно возвращает управление, если порт не содержит ожидающего сервера в момент вызова (неблокирующий запрос).

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_MSG_PORT_INVALID_OBJ	Некорректный объект порта.
FX_MSG_PORT_INVALID_BUF	Некорректные указатели на буфер или переменную для хранения размера ответа.
FX_THREAD_WAIT_TIMEOUT	Нет ожидающих серверов и вызов неблокирующий.
FX_THREAD_WAIT_INTERRUPTED	Выполнение функции прервано сигналом.

Ремарки

Если порт был создан с флагом FX_MSG_PORT_HEADER, тогда пересылаемое сообщение должно иметь зарезервированное место для заголовка. Заголовок имеет тип ker_msg_header_t и зависит от конфигурации ядра.

fx_msg_send_buf

int fx_msg_send_buf(
fx_msg_port_t port, fx_msg_buffer_t* vect, size_t vect_sz, bool block);*

Описание

Отправка в порт сообщения, состоящего из нескольких буферов. Вызывающий поток блокируется тех пор, пока другой поток не получит сообщение из этого порта не ответит на него.

Аргументы

Аргумент	Описание
port	Указатель на объект "порт", который должен быть предварительно открыт. Порт должен быть открыт единожды любым потоком, выполняющейся в контексте данного процесса, после этого все потоки могут посылать в порт сообщения.
vect	Массив структур имеющих тип fx_msg_buffer_t и описывающих буфер сообщения. Тип fx_msg_buffer_t определен как: **typedef struct _fx_msg_buffer_t** **{** **void** buf;* **size_t buf_sz;** **size_t req_sz;** **size_t reply_sz;** **}** **fx_msg_buffer_t;.** buf - Указатель на буфер. buf_sz - Размер буфера (максимальный размер ответа, который может принять данный поток). req_sz - Размер запроса. Если размер запроса меньше, чем размер ожидаемого ответа, это значение меньше, чем buf_sz. reply_sz - Фактический размер ответа сервера. Это число не превышает размера буфера.
vect_sz	Количество буферов в массиве vect. В текущей версии ядра поддерживается отправка только одного или двух буферов.
block	Если это значение равно false, функция немедленно возвращает управление, если порт не содержит ожидающего сервера в момент вызова (неблокирующий запрос).

Возвращаемое значение

Возвращаемое значение	Описание
FX_MSG_PORT_OK	Успешное завершение.
FX_MSG_PORT_INVALID_OBJ	Некорректный объект порта.
FX_MSG_PORT_INVALID_BUF	Некорректные указатели на буфер или переменную для хранения размера ответа.
FX_THREAD_WAIT_TIMEOUT	Нет ожидающих серверов и вызов неблокирующий.
FX_THREAD_WAIT_INTERRUPTED	Выполнение функции прервано сигналом.

Ремарки

Если порт был создан с флагом FX_MSG_PORT_HEADER, тогда пересылаемое сообщение должно иметь зарезервированное место для заголовка. Заголовок имеет тип ker_msg_header_t и зависит от конфигурации ядра. Заголовок всегда пересылается в буфере 0.