Пособие по курсу

Системы реального времени

Справочник по функциям операционной системы реального времени microC/OS-II вер.2.86

Перевод с английского А.П.Новицкого

Технический редактор А.П.Новицкий

Санкт Петербург, 2014

Введение

Этот документ содержит справочную информацию по системным сервисам (функциям) операционной системы uC/OS-II версии 2.86. Все функции, которые предназначены для использования прикладным программистом, перечислены в алфавитном порядке. При подготовке первого варианта данного справочника, в качестве основного источника была использована книга MicroC/OS-II. Real Time Kernel. Second Edition, Jene J. Labrosse, 2002, CMP Books, в которой подробно описано внутреннее устройство uC/OS-II версии 2.60. Однако с тех пор вышло несколько обновлений ОС.

В данном справочнике отражено состояние, соответствующее версии 2.86, датированное 2008.09.12. В этой версии, по сравнению с 2.60 появилось достаточно много новых сервисов. Кроме того, в оригинальном справочнике по функциям при переводе было выявлено значительное количество ошибок, опечаток и неточностей, и для лучшего понимания особенностей сервисов в ходе технического редактирования было сделано некоторое количество пояснений и замечаний.

Обратите внимание на то, что в других документах, а также в исходных текстах ОС вы можете встретить функции, предназначенные для внутреннего использования операционной системой. Эти функции не рекомендованы для использования в прикладных программах, так как их вызов из пользовательского приложения может привести к нежелательным последствиям.

Для каждой функции в справочнике дается следующая информация:

- Краткое описание действия функции.
- Имя файла ОС, содержащее исходный код функции.
- Константа, определяемая в конфигурационном файле для разрешения трансляции кода описываемой функции.
- Прототип функции.
- Описание аргументов, передаваемых функции.
- Описание возвращаемого функцией значения.
- Замечания по особенностям использования предоставляемого сервиса.
- Примеры использования.

OS_ENTER_CRITICAL(), OS_EXIT_CRITICAL()

Описание – Сервисы OS_ENTER_CRITICAL() и позволяют организовать *критическую секцию кода* (участок программы, при выполнении которого запрещены прерывания процессору.
OS_ENTER_CRITICAL() служит для запрета прерывания процессору (вход в *критическую секцию*). Поскольку состояние процессора перед выполнением OS_ENTER_CRITICAL() может быть любым (прерывания уже могли быть запрещены ранее), OS_ENTER_CRITICAL() должен перед запретом прерывания запомнить это состояние.
Сервис OS_EXIT_CRITICAL() восстанавливает состояние запрета/разрешения прерывания, которое имело место перед выполнением ближайшего предыдущего OS_ENTER_CRITICAL(), т.е. завершает *критическую секцию*.

Файл - os_cpu.h

Вызов – из *Задачи* или из *Обработчика прерывания*

Транслируется всегда.

Аргументы – нет.

Возвращаемое значение – нет.

Замечания 1)Данные сервисы реализованы как макросы. Это облегчает портирование ОС. Они всегда должны использоваться «в паре», иными словами, количество входов в *критическую секцию* должно быть равно количеству выходов из *критической секции*.

Если константа OS_CRITICAL_METHOD (см. файл \PORT\os_cpu.h) имеет значение 3, предполагается, что функция, вызывающая эти макросы,выделяет локальную переменную типа OS_CPU_SR с предопределенным именем cpu_sr для хранения содержимого регистра состояния процессора, как показано в следующем фрагменте кода:

```
#if OS_CRITICAL_METHOD == 3 /* Allocate storage for CPU status reg. */
OS_CPU_SR cpu_sr;
```

О методах организации критических секций см. разд.??? книги ???.
В uC/OS-II версий 2.60 и выше предполагается, что используется только OS_CRITICAL_METHOD==3.

Пример использования.

```
void TaskX(void *p_arg) {
  #if OS_CRITICAL_METHOD == 3   //  Поскольку всегда используется метод 3...
    OS_CPU_SR cpu_sr = 0;
  #endif            // ... директива условной трансляции может отсутствовать
  for (;;) {
       ... ...
    OS_ENTER_CRITICAL(); /* Запретить прерывания процессору */
      ... ...     /* Выполнять код критической секции */
    OS_EXIT_CRITICAL();
                /* Восстановить состояние до входа в критическую секцию */
      ... ...
  }
}
```

OSEventNameGet()

Описание - Функция позволяет получить имя объекта: семафора, мьютекса, почтового ящика или очереди сообщений. Имя задается ASCII–строкой, которая может содержать до `OS_EVENT_NAME_SIZE` символов, включая NUL-терминатор. Определена впервые в версии 2.60. Эта функция в основном предназначена для отладочных целей.

Файл - os_core.c

Вызов – только из *Задачи*

Транслируется, если `OS_EVENT_NAME_SIZE>0`.

Прототип INT8U OSEventNameGet(OS_EVENT *pevent,
INT8U *pname,
INT8U *perr);

Аргументы:

`pevent` указатель на блок управления событием. Указатель `pevent` может указывать на следующие объекты синхронизации или коммуникации: *семафор, мьютекс, почтовый ящик или очередь сообщений*. При использовании функции получения имени, тип объекта безразличен. Указатель `pevent` ваша программа получает при создании объекта (см. описания функций `OSSemCreate()`, `OSMutexCreate()`, `OSMboxCreate()`, `OSQCreate()`)

`pname` указатель на `ASCII`-строку, в которую будет возвращено имя объекта. Размер строки должен быть не меньше, чем `OS_EVENT_NAME_SIZE` байтов.

`perr` указатель на переменную, в которую функция возвратит код завершения (ошибки). Этот код может иметь одно из следующих значений:

`OS_ERR_NONE`	если имя объекта было успешно скопировано в переменную по адресу `pname`;
`OS_ERR_EVENT_TYPE`	если указатель `pevent` указывает НЕ на *семафор, мьютекс, почтовый ящик* или *очередь сообщений*;
`OS_ERR_PEVENT_NULL`	если передан нулевой указатель `pevent`
`OS_ERR_NAME_GET_ISR`	если функция вызвана из обработчика прерывания.

Возвращаемое значение – длина `ASCII`-строки, скопированной по адресу `pname`.

Замечания – объект, имя которого требуется получить, должен быть создан ДО того, как будет использована данная функция.

Пример использования.

```
INT8U PrinterSemName[30];
OS_EVENT *PrinterSem;

void Task (void *p_arg)  {
  INT8U err;
  INT8U size;

  for (;;) {
       ...
    size = OSEventNameGet(PrinterSem, &PrinterSemName[0], &err);
       ...
  }
}
```

OSEventNameSet()

Описание – Функция позволяет задать имя объекту: *семафору*, *мьютексу*, *почтовому ящику* или *очереди сообщений*. Имя задается ASCII – строкой, которая может содержать до OS_EVENT_NAME_SIZE символов, включая NULL-терминатор. Определена впервые в версии 2.60.

Файл - os_core.c

Вызов – только из *Задачи*.

Транслируется, если OS_EVENT_NAME_SIZE>0. ??? или >1 ?

Прототип void OSEventNameSet(OS_EVENT *pevent,
INT8U *pname,
INT8U *perr);

Аргументы:

pevent указатель на *Блок управления событием*, pevent может указывать на следующие объекты синхронизации или коммуникации: *семафор*, *мьютекс*, *почтовый ящик* или *очередь сообщений*. При использовании функции задания имени, тип объекта безразличен. Указатель pevent ваша программа получает при создании объекта (смотри описания функций OSSemCreate(), OSMutexCreate(), OSMboxCreate(), OSQCreate())

pname указатель на ASCII строку, которая содержит имя объекта. Размер строки должен быть не больше, чем OS_EVENT_NAME_SIZE байтов, включая NULL-терминатор.

perr указатель на переменную, в которую функция возвратит код завершения (ошибки). Этот код может иметь одно из следующих значений:

OS_ERR_NONE	если имя объекта было успешно задано объекту из переменной по адресу pname;
OS_ERR_EVENT_TYPE	если указатель указывает НЕ на *семафор*, *мьютекс*, *почтовый ящик* или *очередь сообщений*;
OS_ERR_PEVENT_NULL	если в pevent передан нулевой указатель;
OS_ERR_NAME_SET_ISR	если функция вызвана из обработчика прерывания.

Возвращаемое значение – нет.

Замечания – объект, имя которого требуется задать, должен быть создан ДО того, как будет использована данная функция.

Пример использования.

```
OS_EVENT *PrinterSem;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
       ...
    OSEventNameSet(PrinterSem, ”Printer #1”, &err);
       ...
  }
}
```

OSEventPendMulti()

Описание – Функция используется, если предполагается, что *Задача* может ожидать одного из нескольких событий. Определена впервые в версии 2.86.

Файл - os_core.c

Вызов – только из *Задачи*.

Транслируется, если OS_EVENT_MULTI_EN > 0.

Прототип INT16U OSEventPendMulti(OS_EVENT **pevents_pend,
OS_EVENT **pevents_rdy,
void **pmsgs_rdy,
INT16U timeout,
INT8U *perr);

Аргументы:

`pevents_pend` указатель на массив указателей типа `OS_EVENT` и заканчивающийся нулевым элементом. Массив должен быть заранее заполнен указателями на те объекты, которых может ожидать вызывающая *Задача*, и которые могут указывать только на следующие объекты: *Семафор*, *Почтовый ящик* или *Очередь сообщений*. Указатели на объекты ваша программа получает при их создании (смотри описания функций `OSSemCreate()`, `OSMboxCreate()`, `OSQCreate()`)

`pevents_rdy` указатель на массив указателей типа `OS_EVENT`. В этот массив функция `OSEventPendMulti()` возвратит указатели на доступные объекты. Размер этого массива должен быть не меньше размера массива `pevents_pend`, включая конечный нулевой элемент.

`pmsgs_rdy` указатель на массив указателей типа , в который функция `OSEventPendMulti()` возвратит сообщения из объектов коммуникации (если таковые есть). Размер этого массива должен быть не меньше размера массива `pevents_pend`, исключая конечный нулевой элемент. Поскольку нулевые указатели (из *Очереди сообщений*) являются допустимыми значениями, для этого массива нельзя использовать конечный нулевой элемент. Индексы элементов массива `pmsgs_rdy` соответствуют индексам массива `pevents_rdy`.??? Сообщения могут быть возвращены только в те элементы `pmsgs_rdy`, которым в `pevents_rdy` соответствуют указатели на *Почтовые ящики* или *Очереди сообщений*. Все прочие элементы массива `pmsgs_rdy` заполняются нулевыми значениями.

`timeout` если в течение указанного количества системных тиков не возникло события, приостановленная *Задача* будет переведена в состояние готовности. Нулевая величина параметра означает неограниченное время ожидания. Максимальная величина тайм-аута составляет 65535 тиков. Начало тайм-аута не синхронизировано с системными тиками, поэтому реальный тайм-аут лежит в пределах (`timeout ... timeout-1`) тиков.

`perr` указатель на переменную, в которую функция возвратит код завершения (ошибки). Этот код может иметь одно из следующих значений:

`OS_ERR_NONE` если возникло одно из ожидаемых событий, для определения события следует проверять содержимое массива `pevents_rdy`;
`OS_ERR_TIMEOUT` в течение указанного тайм-аута не возникло событий;
`OS_ERR_PEND_ABORT` ожидание событий было отменено функцией ...Abort; для проверки того, какие это были события, следует проверять содержимое массива `pevents_rdy`;
`OS_ERR_EVENT_TYPE` если указатель `pevents_pend` указываетна массив, в котором содержатся НЕ только указатели на *семафор*, *почтовый ящик* или *очередь сообщений*;
`OS_ERR_PEND_LOCKED` если функция `OSEventPendMulti()` вызвана при запрещенной диспетчеризации
`OS_ERR_PEND_ISR` если функция вызвана из обработчика прерывания
`OS_ERR_PEVENT_NULL` если хотя бы один из первых трех параметров есть нулевой указатель.

Возвращаемое значение – нет.

Замечания – объект, имя которого требуется задать, должен быть создан ДО того, как будет использована данная функция.

Пример использования.

???

OSFlagAccept()

Описание – Функция позволяет проверить состояние комбинации битов (установлены или сброшены) в *группе флагов событий*. Проверку можно производить по условию «все» либо по условию «хотя бы один». Данная функция работает так же, как функция `OSFlagPend()`, за тем исключением, что она не блокирует (не переводит в состояние *waiting*) вызывающую *задачу*, если указанное условие (состояние флагов) не выполнено.

Файл - os_flag.c

Вызов – из задачи или из обработчика прерывания

Транслируется, если OS_FLAG_EN && OS_FLAG_ACCEPT_EN > 0.

Прототип OS_FLAGS OSFlagAccept(OS_FLAG_GRP *pgrp,
OS_FLAGS flags,
INT8U wait_type,
INT8U *perr);

Аргументы:

`pgrp` указатель на группу флагов событий. Этот указатель возвращает функция создания группы флагов `OSFlagCreate()`.

`flags` битовая маска, показывающая единичными значениями, какие позиции в группе флагов событий следует проверять.

`wait_type` задает, какое условие следует проверять для выбранных позиций в группе флагов. Можно выбрать следующие значения:
`OS_FLAG_WAIT_CLR_ALL` «все нули»
`OS_FLAG_WAIT_CLR_ANY` «хотя бы один нуль»
`OS_FLAG_WAIT_SET_ALL` «все единицы»
`OS_FLAG_WAIT_SET_ANY` «хотя бы одна единица».

Можно добавить к значению `wait_type` опцию `OS_FLAG_CONSUME`, которая приведет к сбросу флагов в группе, если заданное условие выполнено. Например, если требуется проверить выбранные позиции в группе флагов на условие «хотя бы одна единица» и при выполнении условия сбросить все установленные флаги ??? группы, используйте значение `OS_FLAG_WAIT_SET_ANY + OS_FLAG_CONSUME`.

`perr` указатель на переменную, в которую функция возвращает код завершения – одно из следующих значений:

`OS_ERR_NONE` – ошибки отсутствуют, присутствует заданная комбинация флагов.

`OS_ERR_EVENT_TYPE` – первый параметр указывает НЕ на группу флагов событий.

`OS_FLAG_ERR_WAIT_TYPE` – задано неверное значение `wait_type` аргумента.

`OS_FLAG_INVALID_PGRP` - первый параметр имеет значение NULL.

`OS_FLAG_ERR_NOT_RDY` – ожидаемая комбинация флагов отсутствует.

Возвращаемое значение – ожидаемая комбинация флагов, либо 0, если ожидаемая комбинация отсутствует или произошла ошибка.

Замечания Группа флагов должна быть создана перед использованием данной функции.

Функция НЕ блокирует *задачу*, если ожидаемая комбинация флагов отсутствует.

ВАЖНО! Возвращаемое значение с версии 2.80 отличается от того, что было в более ранних версиях. Теперь, вместо слова состояния флагов возвращается состояние ожидаемых вызывающей *задачей* флагов, если такие имеются.

Пример использования.

```
#define ENGINE_OIL_PRES_OK 0x01
#define ENGINE_OIL_TEMP_OK 0x02
#define ENGINE_START 0x04
OS_FLAG_GRP *EngineStatus;

void Task (void *p_arg) {
  INT8U err;
  OS_FLAGS value;
  (void)p_arg;
  for (;;) {
    value = OSFlagAccept(EngineStatus,
    ENGINE_OIL_PRES_OK + ENGINE_OIL_TEMP_OK,
    OS_FLAG_WAIT_SET_ALL,
    &err);
    switch (err) {
    case OS_ERR_NONE:                   // Desired flags are available
       break;
    case OS_FLAG_ERR_NOT_RDY:     // The desired flags are NOT available
break;
    }
  }
}
```

OSFlagCreate()

Описание – функция используется для создания и инициализации *группы флагов* событий.

Файл - os_flag.c

Вызов – из *задачи* или из *startup*-кода.

Транслируется, если OS_FLAG_EN>0.

Прототип OS_FLAG_GRP *OSFlagCreate(OS_FLAGS flags,
INT8U *perr);

Аргументы:

flags – начальное значение, которым будет проинициализирована создаваемая группа флагов.

perr – указатель на переменную, в которую функция вернет код завершения, одно из следующих значений:

OS_ERR_NONE, если группа флагов успешно создана.

OS_ERR_CREATE_ISR, если сделана попытка вызова функции из обработчика прерывания.

OS_FLAG_GRP_DEPLETED, если более нет свободных групп флагов. Для увеличения количества групп флагов следует увеличить значение OS_MAX_FLAGS в os_cfg.h.

Возвращаемое значение – указатель на созданную группу флагов событий, если была свободная структура типа OS_FLAG_GRP, в противном случае значение NULL.

Замечания: 1) Группа флагов событий должна быть создана перед любым ее использованием.

Пример использования.

```
OS_FLAG_GRP *EngineStatus;
void main (void) {
  INT8U err;
  OSInit(); /* Initialize µC/OS-II */

  EngineStatus = OSFlagCreate(0x00, &err);     // Create a flag group

  OSStart(); /* Start Multitasking */
}
```

OSFlagDel()

Описание – функция используется для того, чтобы удалить группу флагов событий. Будьте осторожны, поскольку несколько *Задач* могут предполагать и использовать одну и ту же группу флагов событий. Безопасным является подход, при котором перед уничтожением группы флагов событий уничтожаются все *Задачи*, которые эту группу использовали.

Файл - os_flag.c

Вызов – только из *Задачи*.

Транслируется, если OS_FLAG_EN>0 и OS_FLAG_DEL_EN>0.

Прототип
```
OS_FLAG_GRP *OSFlagDel(    OS_FLAG_GRP *pgrp,
                           INT8U opt,
                           INT8U *perr);
```

Аргументы:

`pgrp` указатель на группу флагов событий. Этот указатель возвращает функция создания группы флагов OSFlagCreate().

opt позволяет задать опцию, при которой уничтожение группы флагов произойдет, если только нет *Задач*, ожидающих события в этой группе (OS_DEL_NO_PEND) или же это уничтожение произойдет независимо от наличия ожидающих *Задач* (OS_DEL_ALWAYS). В последнем случае все ожидающие *Задачи* будут переведены в состояние *Ready*.

`perr` указатель на переменную, в которую функция вернет код завершения, одно из следующих значений:

`OS_ERR_NONE`,	если удаление группы флагов *событий* завершилось успешно.
`OS_ERR_DEL_ISR`,	если сделана попытка удалить *группу флагов* из обработчика прерываний.
`OS_FLAG_INVALID_PGRP`,	если в первом параметре передается нулевой указатель.
`OS_ERR_EVENT_TYPE`	если первый параметр указывает НЕ на *группу флагов*.
`OS_ERR_INVALID_OPT`	если во втором параметре указана недопустимая опция.
`OS_ERR_TASK_WAITING`,	если одна или несколько *Задач* ожидают данную *группу флагов*, а во втором параметре указана опция `OS_DEL_NO_PEND`.

Возвращаемое значение – нулевой указатель, если *группа флагов* успешно удалена, либо значение параметра `pgrp`, если удаления не произошло. В последнем случае следует проверить возвращенный код ошибки, чтобы выяснить причину.

Замечания

1) Используйте эту функцию с осторожностью, поскольку другие *Задачи* могут ожидать указанной *группы флагов*.

2) Вызов этой функции может потенциально вызвать запрет прерываний на продолжительное время, которое прямо пропорционально числу *задач*, ожидающих указанную *группу флагов*.

Пример использования.

```
OS_FLAG_GRP *EngineStatusFlags;
void Task (void *p_arg) {
  INT8U err;
  OS_FLAG_GRP *pgrp;
  while (1) {
       ...
    pgrp = OSFlagDel(EngineStatusFlags, OS_DEL_ALWAYS, &err);
    if (pgrp == (OS_FLAG_GRP *)0) {
       /* The event flag group was deleted */
    }
```

```
        . . .
    }
}
```

OSFlagNameGet()

Описание – Функция позволяет получить имя *группы флагов*, которое ей ранее было присвоено функцией `OSFlagNameSet()`. Имя представляет собой ASCII – строку, содержащую до символов, включая `NULL`–терминатор.

Файл - os_flag.c

Вызов – из задачи или из обработчика прерывания.

Транслируется, если определена константа `OS_FLAG_NAME_SIZE`.

Прототип `INT8U OSFlagNameGet( OS_FLAG_GRP *pgrp, INT8U *pname, INT8U *perr);`

Аргументы:

`pgrp` указатель на группу флагов.

`pname` указатель на ASCII-строку, в которую будет помещено имя группы флагов. Длина строки должна быть достаточной, чтобы вместить по меньшей мере `OS_FLAG_NAME_SIZE` символов, включая `NULL`-терминатор.

`perr` указатель на переменную, в которую будет возвращен код завершения, одно из следующих значений:

`OS_ERR_NONE` если имя успешно скопировано в `pname[]`

`OS_ERR_EVENT_TYPE`, если первый параметр указывает не на *группу флагов*.

`OS_ERR_PNAME_NULL`, если второй параметр содержит нулевой указатель.

`OS_ERR_INVALID_PGRP`, если первый параметр содержит нулевой указатель.

Возвращаемое значение – размер ASCII строки, помещенной в массив, на который указывает `pname`, либо 0, если при выполнении функции произошла ошибка.

Замечания 1) Группа флагов событий, имя которой хотим получить, должна быть создана до вызова данной функции.

Пример использования.

```
INT8U EngineStatusName[30];
OS_FLAG_GRP *EngineStatusFlags;

void Task (void *p_arg) {
  INT8U err;
  INT8U size;
  (void)p_arg;
  for (;;) {

    size = OSFlagNameGet(EngineStatusFlags,
                         &EngineStatusName[0],
                         &err);
       ...
  }
}
```

OSFlagNameSet()

Описание – OSFlagNameSet() позволяет назначить имя группе флагов событий. Имя представляет собой ASCII-строку, которая может содержать до OS_FLAG_NAME_SIZE символов, включая NULL-терминатор.

Файл - os_flag.c

Вызов – из задачи или из обработчика прерывания

Транслируется, если OS_EVENT_NAME_SIZE>0.

Прототип void OSFlagNameSet(OS_FLAG_GRP *pgrp,
char *pname,
INT8U *perr);

Аргументы:

pgrp указатель на группу флагов событий, которой требуется назначить имя. Этот указатель возвращает функция создания группы флагов OSFlagCreate().

pname указатель на ASCII строку, которая содержит назначаемое имя. Размер строки не должен превышать величины OS_EVENT_NAME_SIZE, включая NULL-терминатор.

perr указатель на переменную, в которую функция вернет код завершения, имеющий одно из следующих значений:

OS_ERR_NONE, если имя группы флагов скопировано в строку по адресу pname.

OS_ERR_EVENT_TYPE, если prgp не указывает на группу флагов.

OS_ERR_PNAME_NULL, если pname имеет значение NULL.

OS_ERR_INVALID_PGRP, если prgp имеет значение NULL.

Возвращаемое значение – нет.

Замечания 1) Группа флагов событий, имя которой хотим задать, должна быть создана до вызова данной функции.

Пример использования.

```
OS_FLAG_GRP *EngineStatus;
void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
       ...
  OSFlagNameSet(EngineStatus, "Engine Status Flags", &err);
       ...
  }
}
```

OSFlagPend()

Описание – функция `OSFlagPend()` позволяет *задаче* проверить состояние комбинации *флагов*. Если условие не выполнено, *задача* переводится в состояние ожидания. Функция позволяет анализировать как условие «все заданные *флаги* установлены» так и условие «хотя бы один из указанных *флагов* установлен».

Файл - os_flag.c

Вызов – только из *Задачи*.

Транслируется, если `OS_FLAG_EN>0`.

Прототип OS_FLAGS OSFlagPend(OS_FLAG_GRP *pgrp,
OS_FLAGS flags,
INT8U wait_type,
INT16U timeout,
INT8U *perr);

Аргументы:

pgrp указатель на *группу флагов* событий. Этот указатель возвращает функция OSFlagCreate() при создании *группы флагов*.

flags битовая маска, в которой «единицы» показывают, какие биты в *группе флагов* следует проверять.

`wait_type` определяет, какое условие следует проверять, одно из следующих значений:

`OS_FLAG_WAIT_CLR_ALL` – все указанные биты равны нулю.

`OS_FLAG_WAIT_CLR_ANY` – хотя бы один из указанных битов равен нулю.

`OS_FLAG_WAIT_SET_ALL` – все указанные биты равны единице.

`OS_FLAG_WAIT_SET_ANY` – хотя бы один из указанных битов равен единице.

`OS_FLAG_CONSUME` – добавление этой опции к параметру `wait_type` позволяет в случае, если заданные условия выполнены, перед завершением функции `OSFlagPend()` сбросить в *группе флагов* все *флаги*, заданные битовой маской `flags`. Например, если задать `wait_type=OS_FLAG_WAIT_SET_ANY+OS_FLAG_CONSUME`, то при установке одного или более из указанных флагов, *задача*, вызвавшая , будет переведена в состояние готовности, а все *флаги*, отмеченные маской, будут сброшены в 0.

`timeout` позволяет *задаче* продолжить выполнение, если указанная комбинация флагов не установилась в течение заданного *тайм-аута* (задаваемого в системных тиках). Максимальное значение *тайм*-аута равно 65535. Значение *тайм-аута*, равное 0 означает бесконечное ожидание заданной комбинации флагов. *Тайм-аут* не синхронизирован с системными тиками, если задано значение 1, то возобновление *задачи* может произойти даже почти немедленно.

`perr` указатель на переменную, в которую функция возвращает код завершения, одно из следующих значений:

`OS_ERR_NONE` отсутствие ошибки, заданная комбинация флагов возникла.

`OS_ERR_PEND_ISR` вызов функции из обработчика прерывания, что недопустимо.

`OS_FLAG_INVALID_PGRP` указатель на группу флагов имеет значение `NULL`.

`OS_ERR_EVENT_TYPE` указатель не указывает на группу флагов.

OS_TIMEOUT истек указанный тайм-аут.

`OS_FLAG_ERR_WAIT_TYPE` аргумент `wait_type` имеет недопустимое значение.

Возвращаемое значение – комбинация флагов, вызвавшая срабатывание, либо 0, если произошла ошибка.

Замечания 1) Группа флагов событий должна быть создана перед ее использованием.

2) Возвращаемое значение в версии 2.80 отличается от предыдущих версий. В более ранних версиях возвращаемое значение содержало текущее состояние флагов, в настоящей и более поздних версиях возвращаемое значение содержит только значения флагов, вызвавшие удовлетворение указанного условия (т.е. коньюнкцию ((`*pgrp)& flags)`).

Пример использования.

```
#define ENGINE_OIL_PRES_OK 0x01
#define ENGINE_OIL_TEMP_OK 0x02
#define ENGINE_START 0x04

OS_FLAG_GRP *EngineStatus;

void Task (void *p_arg) {
  INT8U err;
  OS_FLAGS value;
  (void)p_arg;
  for (;;) {
    value = OSFlagPend(    EngineStatus,
                           ENGINE_OIL_PRES_OK + ENGINE_OIL_TEMP_OK,
                           OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUME,
                           10,
                           &err);
    switch (err) {
      case OS_ERR_NONE:    // Desired flags are available
        break;
      case OS_TIMEOUT:     // The desired flags were NOT available
                           // before 10 ticks occurred
        reak;
    }
  }
}
```

OSFlagPendGetFlagsRdy()

Описание – функция `OSFlagPendGetFlagsRdy()` позволяет *задаче* проверить, какие *флаги* вызвали переход *задачи* в состояние готовности к выполнению. Это имеет смысл при задании условия «хотя бы один» - функция позволяет узнать, которые из *флагов* установились.

Вызов – только из *Задачи*

Транслируется, если `OS_FLAG_EN>0`.

Прототип OS_FLAGS OSFlagPendGetFlagsRdy(void).

Аргументы – нет.

Возвращаемое значение – значения *флагов*, вызвавшие переход текущей *Задачи* в состояние *готовности* к исполнению.

Замечания 1) *группа флагов* должна быть создана до того, как будет использоваться.

Пример использования.

```
#define ENGINE_OIL_PRES_OK 0x01
#define ENGINE_OIL_TEMP_OK 0x02
#define ENGINE_START 0x04
OS_FLAG_GRP *EngineStatus;

void Task (void *p_arg) {
  INT8U err;
  OS_FLAGS value;
   (void)p_arg;
  for (;;) {
    value = OSFlagPend(    EngineStatus,
                           ENGINE_OIL_PRES_OK + ENGINE_OIL_TEMP_OK,
                           OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUME,
                           10,
                           &err);
    switch (err) {
      case OS_ERR_NONE:    // Find out who made task ready
        flags = OSFlagPendGetFlagsRdy();
        break;
      case OS_TIMEOUT:     /* The desired flags were NOT available before .. */
                           /* .. 10 ticks occurred */
        break;
    }
  }
}
```

OSFlagPost()

Описание – функция `OSFlagPost()` устанавливает или сбрасывает биты *флагов* в соответствии с заданной битовой маской `flags` и параметром `opt`, задающим установку либо сброс *флагов*.

Файл - `os_flag.c`.

Вызов – из задачи или из обработчика прерывания.

Транслируется, если `OS_FLAG_EN>0`.

Прототип OS_FLAGS OSFlagPost(OS_FLAG_GRP *pgrp,
OS_FLAGS flags,
INT8U opt,
INT8U *perr);

Аргументы:

`pgrp` указатель на *группу флагов* событий. Этот указатель возвращает функция `OSFlagCreate()` при создании *группы флагов*.

`flags` маска, в которой «единицы» показывают, какие биты в *группе флагов* следует установить или сбросить. Если `opt` равно `OS_FLAG_SET`, каждый бит, который установлен в `flags`, вызовет установку соответствующего бита в *группе флагов*. Если `opt` равно `OS_FLAG_CLR`, каждый бит, который установлен в `flags`, вызовет сброс соответствующего бита в *группе флагов*.

`opt` Задает, будут ли *флаги* устанавливаться (`OS_FLAG_SET`) или сбрасываться (`OS_FLAG_CLR`).

`perr` указатель на переменную, в которую функция возвращает код завершения, один из следующих:

`OS_ERR_NONE`	*флаги* изменены успешно.
`OS_FLAG_INVALID_PGRP`	указатель на *группу флагов* имеет значение `NULL`.
`OS_ERR_EVENT_TYPE`	указатель не указывает на *группу флагов*.
`OS_FLAG_INVALID_OPT`	аргумент `opt` имеет недопустимое значение.

Возвращаемое значение – новое состояние *флагов*.

Замечания 1) *Группа флагов* событий должна быть создана перед использованием.

2) Время выполнения этой функции зависит от количества *задач*, ожидающих данную *группу флагов*, однако, несмотря на это, время выполнения может быть точно оценено.

3) Длительность интервала времени, в течение которого прерывания запрещены, также зависит от количества *задач*, ожидающих данную *группу флагов*.

Пример использования.

```
#define ENGINE_OIL_PRES_OK 0x01
#define ENGINE_OIL_TEMP_OK 0x02
#define ENGINE_START 0x04
OS_FLAG_GRP *EngineStatusFlags;

void TaskX (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
       ...
    err = OSFlagPost(      EngineStatusFlags,
                           ENGINE_START,
                           OS_FLAG_SET,
                           &err);
       ...
  }
```

```
}
```

OSFlagQuery()

Описание – функция используется для получения текущего состояния группы флагов событий. Пока данная функция не позволяет получить список *задач*, ожидающих данную группу событий.

Файл - os_flag.c

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_FLAG_EN && OS_FLAG_QUERY_EN>0.

Прототип `OS_FLAGS OSFlagQuery( OS_FLAG_GRP *pgrp, INT8U *perr);`

Аргументы:

pgrp указатель на *группу флагов* событий. Этот указатель возвращает функция OSFlagCreate() при создании *группы флагов*..

`perr` указатель на переменную, в которую будет помещен код завершения, одно из следующих значений:

`OS_ERR_NONE`	успешное завершение функции.
`OS_FLAG_INVALID_PGRP`	указатель на *группу флагов* `pgrp` имеет значение `NULL`.
`OS_ERR_EVENT_TYPE`	указатель `pgrp` не указывает на *группу флагов*.

Возвращаемое значение – текущее состояние *группы флагов* событий.

Замечания 1) *Группа флагов* событий должна быть создана до ее использования.

2) Данную функцию можно вызывать из обработчика прерываний.

Пример использования.

```
OS_FLAG_GRP *EngineStatusFlags;

void Task (void *p_arg) {
  OS_FLAGS flags;
  INT8U err;
  for (;;) {
       ...
    flags = OSFlagQuery(EngineStatusFlags, &err);
       ...
  }
}
```

OSInit()

Описание – функция инициализирует внутренние структуры данных ОС и должна вызываться до вызова функции `OSStart()`, которая запускает диспетчер *Задач*.

Файл - os_core.c

Вызов – только из функции `main()`.

Транслируется – всегда.

Прототип void OSInit(void);

Аргументы – нет.

Возвращаемое значение – нет.

Замечания – нет.

Пример использования.

```
void main (void) {
       ...
OSInit(); // Инициализация µC/OS-II
       ...
OSStart(); // Запуск многозадачности
}
```

OSIntEnter()

Описание – функция уведомляет ядро о входе в обработчик прерывания, что позволяет µC/OS-II отслеживать уровень вложенности прерываний. Данная функция должна использоваться «в паре» с функцией `OSIntExit()`.

Файл – OS_CORE.C

Вызов – только из обработчика прерывания.

Транслируется - всегда.

Прототип void OSIntEnter(void);

Аргументы – нет.

Возвращаемое значение – нет.

Замечания 1) Эта функция не должна вызываться с уровня *Задачи*.

2) Можно вместо вызова этой функции в пользовательском обработчике прерывания напрямую инкрементировать счетчик вложенности прерываний – переменную `OSIntNesting`. Это позволит уменьшить накладные расходы на вызов и возврат. Выполнять инкремент указанной переменной безопасно, поскольку предполагается, что в этот момент прерывания запрещены (в обработчике прерываний).

3) Глубина вложенности прерываний не должна превышать 255.

OSIntExit()

Описание – функция уведомляет ядро о том, что выполнение обработчика прерывания завершено, что позволяет ОС отслеживать вложенность прерываний. Функция `OSIntExit()` должна использоваться в паре с функцией `OSIntEnter()`. Когда последний в цепи вложенных обработчиков завершается (и вложенность становится равной нулю), функция `OSIntExit()` определяет *Задачу* с наивысшим приоритетом, готовую к выполнению, и передает ей управление.

Файл - `os_core.c`.

Вызов – только из обработчика прерывания.

Транслируется всегда.

Прототип void OSIntExit(void);

Аргументы – нет.

Возвращаемое значение – нет.

Замечания 1) данная функция не должна вызываться с уровня *Задач*. В отличие от функции `OSIntEnter()`, функцию `OSIntExit()` нельзя просто заменить оператором, декрементирующим счетчик вложенности `OSIntNesting`, следует обязательно вызывать `OSIntExit()`.

OSMboxAccept()

Описание – функция позволяет проверить, имеется ли сообщение в *Почтовом ящике*, указатель на который передается в качестве параметра. Функция `OSMboxAccept()` в отличие от функции `OSMboxPend()`, не блокирует вызвавшую *Задачу*, если сообщение отсутствует. Если сообщение имеется, оно немедленно возвращается вызвавшей *Задаче*, а *Почтовый ящик* очищается. Функцию `OSMboxAccept()` используют обычно в обработчике прерывания, поскольку обработчик нельзя блокировать (перевести в состояние ожидания).

Файл - `os_mbox.c`.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_MBOX_EN && OS_MBOX_ACCEPT_EN>0.

Прототип void *OSMboxAccept(OS_EVENT *pevent);

Аргументы:

`pevent` указатель на *Почтовый ящик*, из которого требуется принять сообщение. Этот указатель возвращает функция создания *Почтового ящика* `OSMboxCreate()`.

Возвращаемое значение – указатель на сообщение, если оно имеется, либо `NULL`, если *Почтовый ящик* пуст.

Замечания 1) *Почтовый ящик* должен быть создан до его использования.

Пример использования.

```
OS_EVENT *CommMbox;

void Task (void *p_arg) {
  void *msg;
  (void)p_arg;
  for (;;) {
    msg = OSMboxAccept(CommMbox); /* Check mailbox for a message */
    if (msg != (void *)0) {  // Message received, process
       ...
    } else {  //* Message not received, do something else
       ...
    }
  }
}
```

OSMboxCreate()

Описание – функция создает и инициализирует *Почтовый ящик*. *Почтовый ящик* позволяет *Задаче* или обработчику прерывания передать переменную размером в указатель (сообщение) одной или нескольким *Задачам*.

Файл - `os_mbox.c`.

Вызов – из *Задачи* или из функции `main()`.

Транслируется, если `OS_MBOX_EN>0`.

Прототип OS_EVENT *OSMboxCreate(void *msg);

Аргументы:

`msg` - используется для инициализации *Почтового ящика*. *Почтовый ящик* считается пустым, если `msg` равен `NULL`. Это позволяет использовать `Mbox` как двоичный семафор.

И, наоборот, *Почтовый ящик* содержит сообщение, если `msg` не равен `NULL`.

Возвращаемое значение – указатель на *Блок управления событием*, выделенный для создаваемого *Почтового ящика*. Если нет свободных *Блоков управления событиями*, функция `OSMboxCreate()` возвратит нулевой указатель.

Замечания 1) *Почтовый ящик* должен быть создан до его использования.

Пример использования.

```
OS_EVENT *CommMbox;

void main (void) {
      ...
  OSInit(); /* Initialize µC/OS-II */
      ...
..CommMbox = OSMboxCreate((void *)0); /* Create COMM mailbox */
..OSStart(); /* Start Multitasking */
}
```

OSMboxDel()

Описание – используется для удаления *почтового ящика*. Используйте эту функцию с осторожностью, так как другие *задачи* могут ожидать сообщения из удаленного *почтового ящика*, либо пытаться к нему обратиться. Наилучший способ избежать проблем – сначала удалить все *задачи*, которые могут обратиться к *ящику*, подлежащему удалению.

Файл - os_mbox.c.

Вызов – только из *Задачи*.

Транслируется, если OS_MBOX_EN && OS_MBOX_DEL_EN > 0

Прототип `OS_EVENT *OSMboxDel( OS_EVENT *pevent, INT8U opt, INT8U *perr);`

Аргументы:

pevent указатель на *Почтовый ящик*. Этот указатель возвращает функция создания *Почтового ящика* OSMboxCreate().

opt определяет, будет ли *почтовый ящик* удален лишь в случае, когда нет ожидающих его *Задач* (OS_DEL_NO_PEND) , либо он будет удален в любом случае (безусловно) (OS_DEL_ALWAYS). В последнем случае все ожидающие *Задачи* переводятся в состояние *готовности* к исполнению.

perr указатель на переменную, в которую будет помещен код завершения, одно из следующих значений:

OS_ERR_NONE - если объект успешно удален.

OS_ERR_DEL_ISR - если сделана попытка удаления объекта из обработчика прерывания.

OS_ERR_INVALID_OPT – если неверно задано значение аргумента opt.

OS_ERR_TASK_WAITING – если одна или более *Задач* ожидают удаляемого объекта.

OS_ERR_EVENT_TYPE – если указатель pevent не указывает на *Почтовый ящик*.

OS_ERR_PEVENT_NULL если pevent имеет значение NULL.

Возвращаемое значение – нулевой указатель, если *Почтовый ящик* успешно удален, либо pevent, если удаления не произошло. В последнем случае следует проанализировать возвращенный в *perr код ошибки.

Замечания 1) Следует использовать функцию с осторожностью, поскольку другие *Задачи* могут предполагать существование стертого объекта.

2) Следует учитывать, что в период времени, когда *ядро* переводит ожидающие сообщения *Задачи* в состояние готовности к исполнению, прерывания запрещены – это может увеличить время реакции на запрос прерывания.

3) *Задача*, вызвавшая OSMboxAccept() , не имеет возможности узнать, что *Почтового ящика* не существует (что он был удален!).

Пример использования.

```
OS_EVENT *DispMbox;

void Task (void *p_arg) {
  INT8U err;
   (void)p_arg;
  while (1) {
       ....
    DispMbox = OSMboxDel(DispMbox, OS_DEL_ALWAYS, &err);
    if (DispMbox == (OS_EVENT *)0) {
      /* Mailbox has been deleted */
    }
       ....
  }
```

```
}
```

OSMboxPend()

Описание – функция вызывается *Задачей*, ожидающей сообщения в данном *Почтовом ящике*. Сообщение может быть послано другой *Задачей* или обработчиком прерывания. Сообщение представляет собой переменную длиной в указатель, значение сообщения специфично для каждого конкретного приложения. Если в момент обращения к функции `OSMboxPend()` сообщение присутствует в *Почтовом ящике, ящик* очищается, управление передается *Задаче,* а сообщение содержится в возвращаемом функцией `OSMboxPend()` значении.

Файл - os_mbox.c

Вызов – только из *Задачи*.

Транслируется, если `OS_MBOX_EN>0`

Прототип `void *OSMboxPend( OS_EVENT *pevent, INT16U timeout, INT8U *perr);`

Аргументы:

`pevent` указатель на *Почтовый ящик*, из которого ожидается *сообщение*. Этот указатель возвращает функция `OSMboxCreate()` при создании *Почтового ящика*.

`timeout` позволяет *задаче* продолжить исполнение, если сообщение не пришло в течение заданного *тайм-аута* (задаваемого в системных тиках). Максимальное значение *тайм-аута* равно 65535. Значение *тайм-аута*, равное 0 означает бесконечное ожидание. *Тайм-аут* не синхронизирован с системными тиками, если задано значение 1, то возобновление *Задачи* может произойти даже почти немедленно.

`perr` указатель на переменную, в которую возвращается код ошибки, имеющий одно из следующих значений:

`OS_ERR_NONE` - если сообщение успешно принято.

`OS_TIMEOUT` - если сообщение не принято в течение заданного тайм-аута.

`OS_ERR_EVENT_TYPE` - если `pevent` не указывает на *почтовый ящик*.

`OS_ERR_PEND_ISR` - если функция вызвана из обработчика прерывания. Такой вызов недопустим, но *ядро* проверяет это. Обработчик прерывания при отсутствии сообщения не приостанавливается.

OS_ERR_PEVENT_NULL - если pevent равно NULL.

Возвращаемое значение – указатель на сообщение, посланное другой *Задачей* или обработчиком прерывания. Если сообщение в *Почтовом ящике* отсутствует, *Задача* переводится в состояние ожидания. Если сообщение не получено в течение заданного *тайм-аута*, *Задача* возвращается в состояние готовности, происходит возврат из функции `OSMboxPend()`, возвращается нулевой указатель, а переменная `*perr` принимает значение `OS_TIMEOUT`.

Замечания 1) *Почтовый ящик* должен быть создан перед его использованием.

2) функцию `OSMboxPend()` нельзя вызывать из обработчика прерывания.

Пример использования.

```
OS_EVENT *CommMbox;
void CommTask(void *p_arg) {
  INT8U err;
  void *msg;
  for (;;) {
       ...
    msg = OSMboxPend(CommMbox, 10, &err);
    if (err == OS_ERR_NONE) {
      /* Code for received message */
    } else {
      /* Code for message not received within timeout */
    }
```

```
        . . .
    }
}
```

OSMboxPendAbort()

Новая функция, появилась в вер.2.84

Описание – функция переводит в состояние готовности *Задачи*, ожидающие сообщения из данного *Почтового ящика*. Эту функцию следует использовать для аварийного возврата ожидающих *Задач* в состояние готовности, как альтернативу нормальному использованию функций `OSMboxPost()` или `OSMboxPostOpt()`.

Файл - `os_mbox.c`.

Вызов – только из *Задачи*.

Транслируется, если OS_MBOX_EN && OS_MBOX_PEND_ABORT_EN > 0.

Прототип INT8U OSMboxPendAbort (OS_EVENT *pevent,
INT8U opt,
INT8U *perr);

Аргументы:

`pevent` указатель на почтовый ящик, в который помещается сообщение. Этот указатель при создании почтового ящика возвращает функция `OSMboxCreate()`.

`opt` тип аварийного завершения:
`OS_PEND_OPT_NONE` - аварийно переводит в состояние готовности лишь одну *Задачу* с наивысшим приоритетом из ожидающих указанного *Почтового Ящика*;
`OS_PEND_OPT_BROADCAST` – переводит в состояние готовности все *Задачи*, ожидающие данного *Почтового Ящика*.

`perr` указатель на переменную, в которую функция вернет код завершения, одно из следующих значений:
`OS_ERR_NONE` если нет *Задач,* ожидающих указанного *Почтового Ящика*;
`OS_ERR_PEND_ABORT` хотя бы одна *Задача*, ожидавшая *Почтового Ящика*, была переведена в состояние готовности, возвращаемое значение содержит число таких *Задач*;
`OS_ERR_EVENT_TYPE` если `pevent` указывает НЕ на *Почтовый Ящик*;
`OS_ERR_PEVENT_NULL` если `pevent` имеет значение `NULL`.

Возвращаемое значение – число *Задач,* переведенных в состояние готовности. Нуль указывает, что таких *Задач* не было.

Замечания 1) *Почтовый ящик* должен быть создан до его использования.

Пример использования:

```
void CommTask(void *p_arg) {
  INT8U err;
  INT8U nbr_tasks;
  (void)p_arg;
  for (;;) {
       ...
    nbr_tasks = OSMboxPendAbort(CommMbox, OS_PEND_OPT_BROADCAST, &err);
    if (err == OS_ERR_NONE) {
      /* No tasks were waiting on the mailbox */
    } else {
      /* All pends of tasks waiting on mailbox were aborted … */
      /* ... 'nbr_tasks' indicates how many were made ready. */
    }
....
  }
}
```

OSMboxPost()

Описание – функция позволяет послать сообщение из *Задачи* или из обработчика прерывания. Сообщение представляет собой переменную длиной в указатель. Если сообщение уже имеется в *Почтовом ящике*, функция возвращает код ошибки, показывающий, что *Почтовый ящик* не пуст. При этом функция `OSMboxPost()` немедленно возвращает управление вызывающей программе, новое *сообщение* НЕ помещается в *Почтовый ящик*. Если какие-либо *Задачи* ожидают сообщений из данного *Почтового ящика*, а функция `OSMboxPost()` поместила в *ящик* новое сообщение, оно передается ожидающей *Задаче*, имеющей наивысший приоритет, а *Почтовый ящик* переходит в состояние «пуст». Затем, если приоритет *Задачи*, принявшей сообщение выше, чем у *Задачи*, пославшей сообщение, принявшая *Задача* возобновляется (т.е. переводится в состояние исполнения), а *Задача*, пославшая сообщение, приостанавливается (переводится в состояние *готовности*).

Файл - `os_mbox.c`.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_MBOX_EN && OS_MBOX_POST_EN > 0.

Прототип INT8U OSMboxPost(OS_EVENT *pevent,
void *msg);

Аргументы:

`pevent` указатель на *Почтовый ящик*, в который помещается сообщение. Этот указатель при создании *Почтового ящика* возвращает функция `OSMboxCreate()`.

`msg` сообщение, посылаемое *Задаче*. Сообщение это переменная длиной в указатель, значение этой переменной задается программистом. Не следует помещать в *Почтовый ящик* значение `NULL`, так как оно означает, что *ящик* пуст.

Возвращаемое значение – одно из следующего перечня значений:

`OS_ERR_NONE` успешное завершение, сообщение помещено в *почтовый ящик*.

`OS_MBOX_FULL` если *почтовый ящик* уже содержит сообщение.

`OS_ERR_EVENT_TYPE` если `pevent` указывает не на *Почтовый ящик*.

OS_ERR_PEVENT_NULL если pevent имеет значение NULL.

`OS_ERR_POST_NULL_PTR` если делается попытка передать в качестве *сообщения* нулевой указатель.

Замечания 1) *Почтовый ящик* должен быть создан до его использования.

2) Не следует передавать в качестве сообщения нулевое значение, поскольку оно означает, что *почтовый ящик* пуст.

Пример использования.

```
OS_EVENT *CommMbox;
INT8U CommRxBuf[100];

void CommTaskRx (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
       ...
    err = OSMboxPost(CommMbox, (void *)&CommRxBuf[0]);
       ...
  }
}
```

OSMboxPostOpt()

Описание – OSMboxPostOpt() работает аналогично функции OSMboxPost(), за исключением того, что позволяет отправить сообщение многим *Задачам*. Иными словами, отправка такого «широковещательного» сообщения сделает готовыми к исполнению все *Задачи*, ожидающие сообщений из данного *Почтового ящика*. Функция OSMboxPostOpt() является расширением функции OSMboxPost() в том смысле, что может эмулировать поведение последней. *Сообщение* представляет собой переменную размером с указатель, значение этой переменной является специфичным для конкретного приложения. Если в *Почтовом ящике* уже имеется *сообщение*, функция OSMboxPostOpt() немедленно завершается, новое *сообщение* НЕ помещается в *Почтовый ящик*, и возвращается код соответствующей ошибки. Если несколько *Задач* ожидают *сообщения*, функция OSMboxPostOpt() позволяет отправить сообщение либо единственной *Задаче*, наиболее приоритетной из ожидающих, для этого opt должен иметь значение OS_POST_OPT_NONE, либо всем *Задачам*, ожидающим данного *почтового ящика*, для этого параметр должен иметь значение OS_POST_OPT_BROADCAST. В любом из этих случаев *ядро* выполнит операцию *планирования*, и если *Задача* с наивысшим приоритетом из получивших сообщение имеет приоритет выше, чем *Задача*, пославшая сообщение, то первая будет возобновлена (Run), а *вторая* - приостановлена (*Ready*), т.е. произойдет переключение контекста (*диспетчеризация*).

Файл - os_mbox.c.

Вызов – из *Задачи* или из обработчика прерывания.

Транслируется, если OS_MBOX_EN && OS_MBOX_POST_OPT_EN > 0

Прототип INT8U OSMboxPostOpt(OS_EVENT *pevent,
void *msg,
INT8U opt);

Аргументы:

pevent указатель на *Почтовый ящик*. Этот указатель возвращает функция OSMboxCreate().

msg сообщение, которое посылается *Задачам*. Параметр msg это переменная размером с указатель, ее значение не специфицировано, а выбирается разработчиком конкретного приложения. Недопустимо передавать нулевое значение, поскольку оно означает отсутствие сообщения.

opt определяет, посылается ли сообщение единственной наиболее приоритетной из ожидающих *задач* (если opt имеет значение OS_POST_OPT_NONE) или всем *Задачам* ожидающим сообщения из заданного *Почтового ящика* (если opt имеет значение OS_POST_OPT_BROADCAST).

Возвращаемое значение – код завершения – одно из следующих значений:

OS_ERR_NONE если вызов завершился без ошибок.

OS_MBOX_FULL если *Почтовый ящик* уже содержит *сообщение*. *Сообщение* должно быть извлечено из *ящика*, для того, чтобы туда можно было поместить следующее.

OS_ERR_EVENT_TYPE если указатель pevent указывает указывает НЕ на *Почтовый ящик*.

OS_ERR_PEVENT_NULL если pevent имеет значение NULL.

OS_ERR_POST_NULL_PTR если сделана попытка передать сообщение, имеющее значение NULL.

Замечания 1) *Почтовый ящик* должен быть создан до его использования.

2) Нельзя передавать сообщение со значением NULL, так как это значение показывает, что *почтовый ящик* пуст.

3) Если необходимо использовать эту функцию и требуется минимизировать размер кода, то можно запретить трансляцию кода функции OSMboxPost(), поскольку OSMboxPostOpt() может эмулировать действие функции OSMboxPost().

Пример использования.

```
OS_EVENT *CommMbox;
INT8U CommRxBuf[100];
void CommRxTask (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
       ...
    err = OSMboxPostOpt(   CommMbox,
                                  (void *)&CommRxBuf[0],
                                  OS_POST_OPT_BROADCAST);
       ...
  }
}
```

OSMboxQuery()

Описание – функция OSMboxQuery() предоставляет информацию о состоянии *Почтового ящика*. Для ее получения следует перед вызовом функции объявить структуру данных типа OS_MBOX_DATA, в которую функция скопирует содержимое ECB, принадлежащего *Почтовому ящику*. Функция позволяет определить, ожидают ли этого *Почтового ящика* какие-либо *Задачи* и сколько этих *Задач* (для этого следует подсчитать количество «единиц» в поле P_mbox_data:OSEventTbl[]. Также можно получить текущее содержимое *Почтового ящика*.

Файл - os_mbox.c.

Вызов – из задачи или из обработчика прерывания

Транслируется , если OS_MBOX_EN && OS_MBOX_QUERY_EN > 0.

Прототип INT8U OSMboxQuery(OS_EVENT *pevent,
OS_MBOX_DATA *p_mbox_data);

Аргументы:

pevent указатель на *Почтовый ящик*. Этот указатель возвращает функция OSMboxCreate() при создании *Почтового ящика*.

P_mbox_data указатель на структуру данных типа OS_MBOX_DATA, которая содержит следующие поля:

```
void *OSMsg; /* Copy of the message stored in the mailbox */
#if OS_VERSION < 280
INT8U OSEventTbl[OS_EVENT_TBL_SIZE]; /* Copy of the mailbox wait list */
INT8U OSEventGrp;
#else
INT16U OSEventTbl[OS_EVENT_TBL_SIZE]; /* Copy of the mailbox wait list */
INT16U OSEventGrp;
#endif
```

Возвращаемое значение – код завершения, который может принимать одно из следующих значений:

OS_ERR_NONE	если вызов функции завершился успешно.
OS_ERR_PEVENT_NULL	если pevent имеет значение NULL.
OS_ERR_EVENT_TYPE	если pevent указывает не на *Почтовый ящик*.
OS_ERR_PNAME_NULL	если p_mbox_data имеет значение NULL.

Замечания 1) *Почтовый ящик* должен быть создан до его использования.

Пример использования.

```
OS_EVENT *CommMbox;
void Task (void *p_arg) {
  OS_MBOXDATA mbox_data;
  INT8U err;
  (void)p_arg;
  for  (;;) {
       ...
    err = OSMboxQuery(CommMbox, &mbox_data);
    if (err == OS_ERR_NONE) {
       /* Mailbox contains a message if mbox_data.OSMsg is not NULL */
    }
       ...
  }
}
```

OSMemCreate()

Описание – функция OSMemCreate создает и инициализирует *раздел* памяти. *Раздел* памяти содержит заданное пользователем количество блоков памяти фиксированного размера. Пользовательское приложение может запросить и получить блок памяти, а когда он станет не нужен, освободить его и вернуть в *раздел*.

Файл - os_mem.c.

Вызов – из *Задачи* или из *starup*-кода.

Транслируется, если OS_MEM_EN > 0.

Прототип
```
OS_MEM *OSMemCreate(       void *addr,
                           INT32U nblks,
                           INT32U blksize,
                           INT8U *perr);
```

Аргументы:

addr начальный адрес области памяти, которая будет использована для создаваемого *раздела* памяти, содержащего блоки фиксированного размера. *Раздел* памяти можно создать как используя статический массив, так и выделив память под *раздел* динамически, с использованием функции malloc() в startup-коде.

nblks задает количество блоков, которые будет содержать создаваемый *раздел* памяти. Следует задавать не менее двух блоков.

blksize задает размер блока (в байтах), который одинаков для всех блоков *раздела*. Размер блока должен быть не меньше размера указателя для используемого процессора (т.е., например, для архитектуры ARM – не менее 4 байтов).

perr указатель на переменную, в которую функция возвратит код завершения, который может иметь одно из следующих значений:

OS_ERR_NONE	если *раздел* памяти создан успешно.
OS_MEM_INVALID_ADDR	если задано неверное значение addr (например NULL).
OS_MEM_INVALID_PART	если нет свободных раздел*ов* памяти.
OS_MEM_INVALID_BLKS	если задано количество блоков, меньшее двух.
OS_MEM_INVALID_SIZE	если задан размер блока меньший, чем размер указателя.

Возвращаемое значение – указатель на блок управления *разделом* памяти, если есть свободный, либо NULL, если свободный блок управления *разделом* памяти отсутствует.

Замечания 1) *Раздел* памяти должен быть создан перед его использованием.

Пример использования.

```
OS_MEM *CommMem;
INT8U CommBuf[16][128];

void main (void) {
  INT8U err;
  OSInit(); /* Initialize µC/OS-II */
       ...
  CommMem = OSMemCreate(&CommBuf[0][0], 16, 128, &err);
       ...
  OSStart(); /* Start Multitasking */
}
```

OSMemGet()

Описание – функция OSMemGet() позволяет выделить *блок* памяти из ранее созданного *раздела*. Предполагается, что вызывающая программа осведомлена о размере запрашиваемого *блока* (это атрибут указываемого в параметрах раздела). После того, как *блок* памяти стал ненужным, его лучше бы освободить с помощью вызова функции OSMemPut().

Файл - os_mem.c

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_MEM_EN>0.

Прототип void *OSMemGet(OS_MEM *pmem,
INT8U *perr);

Аргументы:

pmem указатель на блок управления *разделом* памяти. Этот указатель возвращает функция OSMemCreate() при создании *раздела*.

perr указатель на переменную, в которую функция OSMemGet() передает код завершения: один из следующих:

OS_ERR_NONE если блок памяти выделен вызвавшей *Задаче*.

OS_MEM_NO_FREE_BLKS если указанный *раздел* памяти не содержит свободных блоков памяти.

OS_MEM_INVALID_PMEM если pmem имеет значение NULL.

Возвращаемое значение – указатель на выделенный блок памяти, если он есть, либо NULL, если нет свободных блоков.

Замечания 1) *Раздел* памяти должен быть создан перед использованием.

Пример использования.

```
OS_MEM *CommMem;

void Task (void *p_arg) {
  INT8U *msg;
  (void)p_arg;
  for (;;) {
       ...
    msg = OSMemGet(CommMem, &err);
    if (msg != (INT8U *)0) {
       /* Memory block allocated, use it. */
    }
       ...
  }
}
```

OSMemNameGet()

Описание – OSMemNameGet() позволяет получить (ранее назначенное функцией OSMemNameSet()) имя раздела памяти. Имя представляет собой ASCII-строку, длина имени может содержать до символов, включая завершающий символ NULL.

Файл - os_mem.c.

Вызов – из задачи или из обработчика прерывания

Транслируется, если OS_MEM_NAME_SIZE > 0.

Прототип INT8U OSMemNameGet(OS_MEM *pmem,
INT8U *pname,
INT8U *perr);

Аргументы:

pmem указатель на *раздел* памяти, имя которого запрашивается.

pname указатель на строку, в которую будет возвращено запрашиваемое имя. Строка должна иметь длину не меньше OS_MEM_NAME_SIZE символов (включая NULL).

perr указатель на переменную, в которую функция вернет код завершения, одно из следующих значений:

OS_ERR_NONE если имя *раздела* памяти было успешно скопировано в строку, на которую указывает pname.

OS_ERR_INVALID_PMEM при попытке передать в pmem нулевой указатель.

OS_ERR_PNAME_NULL при попытке передать в pname нулевой указатель.

Возвращаемое значение – размер ASCII-строки, помещенной в массив по адресу pname, либо 0, если при выполнении функции произошла ошибка.

Замечания 1) *раздел* памяти должен быть создан перед его использованием.

Пример использования.

```
OS_MEM *CommMem;
INT8U CommMemName[OS_MEM_NAME_SIZE];

void Task (void *pdata) {
  INT8U err;
  INT8U size;
  for (;;) {
       ...
    size = OSMemNameGet(CommMem, & CommMemName [0], &err);
       ...
  }
}
```

OSMemNameSet()

Описание – OSMemNameSet() позволяет задать имя *разделу* памяти. *Имя* представляет собой ASCII-строку, которая может содержать до OS_MEM_NAME_SIZE символов, включая NULL-терминатор.

Файл - os_mem.c.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_MEM_NAME_SIZE > 0.

Прототип void OSMemNameSet(OS_MEM *pmem,
INT8U *pname,
INT8U *perr);

Аргументы:

pmem указатель на раздел памяти, которому дается имя. Этот указатель возвращает функция OSMemCreate() при создании *раздела.*

pname указатель на ASCII-строку, которая содержит имя, присваиваемое *разделу* памяти. Размер строки не должен превышать OS_MEM_NAME_SIZE символов, включая ограничитель строки NULL.

perr указатель на строку, в которую функция вернет код завершения, одно из следующих значений:

OS_ERR_NONE если имя *раздела* памяти успешно скопировано из массива по адресу pname.

OS_MEM_INVALID_PMEM попытка передать NULL в параметр pmem.

OS_ERR_PNAME_NULL попытка передать NULL в параметр pname.

OS_MEM_NAME_TOO_LONG если имя не умещается в массив по адресу pname.

Возвращаемое значение – отсутствует.

Замечания 1) *раздел* памяти должен быть создан перед любым его использованием.

Пример использования.

```
OS_MEM *CommMem;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
       ...
    OSMemNameSet(CommMem, "Comm. Buffer", &err);
       ...
  }
}
```

OSMemPut()

Описание – функция освобождает блок памяти и возвращает его в *раздел* памяти. Предполагается, что блок корректно возвращается в тот *раздел*, из которого он был получен.

Файл - os_mem.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_MEM_EN > 0.

Прототип INT8U OSMemPut(OS_MEM *pmem,
void *pblk);

Аргументы:

pmem указатель на блок управления *разделом* памяти, этот указатель возвращает функция OSMemCreate() при создании *раздела*.

pblk указатель на блок памяти, который требуется освободить.

Возвращаемое значение – код завершения, который может принимать следующие значения:

OS_ERR_NONE если блок памяти был освобожден успешно.

OS_MEM_FULL *раздел* памяти не может «принять» освобождаемый блок. Это явное свидетельство, что в программе что-то не ладно, так как делается попытка «вернуть» блоков больше, чем до того было получено с помощью OSMemGet().

OS_MEM_INVALID_PMEM если указатель pmem имеет значение NULL.

OS_MEM_INVALID_PBLK если указатель pblk имеет значение NULL.

Замечания 1) *Раздел* памяти должен быть создан перед любым использованием.

2) Следует возвращать блок памяти именно в тот *раздел*, из которого он был получен.

Пример использования.

```
OS_MEM *CommMem;
INT8U *CommMsg;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
       ...
    err = OSMemPut(CommMem, (void *)CommMsg);
    if (err == OS_ERR_NONE) {
    /* Memory block released */
    }
       ...
  }
}
```

OSMemQuery()

Описание – позволяет получить информацию о *разделе* памяти. Эта функция возвращает ту же информацию, что содержится в структуре данных OS_MEM, путем копирования этой информации в новую структуру, называемую OS_MEM_DATA.

Файл - os_mem.c.

Вызов – из задачи или из обработчика прерывания.

Транслируется, если OS_MEM_EN && OS_MEM_QUERY_EN > 0.

Прототип `INT8U OSMemQuery( OS_MEM *pmem, OS_MEM_DATA *p_mem_data);`

Аргументы:

pmem указатель на блок управления *разделом* памяти, этот указатель возвращает функция OSMemCreate() при создании *раздела*.

P_mem_data указатель на структуру данных типа OS_MEM_DATA, которая содержит следующие поля:

```
void *OSAddr;       /* Указатель на начальный адрес раздела памяти       */
void *OSFreeList; /* Указатель на начало списка свободных блоков   */
INT32U OSBlkSize; /* Размер блока в байтах                         */
INT32U OSNBlks;     /* Общее количество блоков в разделе памяти    */
INT32U OSNFree;     /* Количество свободных блоков в разделе             */
INT32U OSNUsed;     /* Количество использованных блоков в разделе  */
```

Возвращаемое значение – код завершения, который может иметь следующие значения:

OS_ERR_NONE	если структура OS_MEM_DATA успешно заполнена.
OS_MEM_INVALID_PMEM	если pmem имеет значение NULL.
OS_MEM_INVALID_PDATA	если p_mem_data имеет значение NULL.

Замечания 1) *Раздел* памяти должен быть создан до его использования.

Пример использования.

```
OS_MEM *CommMem;

void Task (void *p_arg) {
  INT8U err;
  OS_MEM_DATA mem_data;
  for (;;) {
       ...
    err = OSMemQuery(CommMem, &mem_data);
       ...
  }
}
```

OSMutexAccept()

Описание – OSMutexAccept() позволяет проверить состояние *мьютекса*, но в отличие от функции OSMutexPend(), не блокирует (не переводит в состояние Waiting) вызвавшую *Задачу*, если *мьютекс* занят.

Файл - os_mutex.c.

Вызов – только из *Задачи*.

Транслируется, если OS_MUTEX_EN > 0.

Прототип `INT8U OSMutexAccept( OS_EVENT *pevent, INT8U *perr);`

Аргументы:

pevent указатель на *мьютекс*, который ограничивает доступ к ресурсу. Функция OSMutexCreate() возвращает этот указатель при создании *мьютекса*.

perr указатель на переменную, в которую функция возвращает код завершения, одно из следующих значений:

OS_ERR_NONE если вызов функции завершился успешно.

OS_ERR_EVENT_TYPE если аргумент pevent указывает не на *мьютекс*.

OS_ERR_PEVENT_NULL если аргумент pevent имеет значение NULL.

OS_ERR_PEND_ISR если сделана попытка вызвать OSMutexAccept() из *обработчика прерывания*.

Возвращаемое значение – если *мьютекс* доступен, функция возвращает 1, если *мьютекс* занят (его захватила другая *Задача*), функция возвращает 0.

Замечания 1) *Мьютекс* должен быть создан перед любым его использованием.

2) Функцию OSMutexAccept() недопустимо вызывать из *обработчика прерывания*.

3) Если мьютекс захвачен *Задачей* с использованием OSMutexAccept(), следует вызывать OSMutexPost() для освобождения *мьютекса*, когда ресурс становится не нужен.

Пример использования.

```
OS_EVENT *DispMutex;

void Task (void *p_arg) {
  INT8U err;
  INT8U value;
  for (;;) {
       ...
    value = OSMutexAccept(DispMutex, &err);
    if (value == 1) {
      /* Resource available, process */
    } else {
      /* Resource NOT available */
    }
       ...
  }
}
```

OSMutexCreate()

Описание – используется для создания и инициализации *мьютекса*. *Мьютекс* используется для организации доступа к разделяемому ресурсу.

Файл - os_mutex.c.

Вызов – из *Задачи* или из `startup`-кода.

Транслируется, если `OS_MUTEX_EN > 0`.

Прототип OS_EVENT *OSMutexCreate(INT8U prio,
INT8U *perr);

Аргументы:

`prio` уровень наследования приоритета `priority inheritance priority (PIP)`, который используется, когда высокоприоритетная *Задача* пытается получить *мьютекс*, захваченный низкоприоритетной *задачей*. В этом случае уровень приоритета низкоприоритетной *Задачи* «поднимается» на уровень `PIP`, пока она не освободит *мьютекс*.

`perr` указатель на переменную, в которую функция заносит код завершения, одно из следующих значений:

`OS_ERR_NONE`	если вызов был успешным, и *мьютекс* создан.
`OS_ERR_CREATE_ISR`	если сделана попытка создать *мьютекс* из *обработчика прерывания*.
`OS_PRIO_EXIST`	если существует *задача* с приоритетом, равным заданному `PIP`.
`OS_ERR_PEVENT_NULL`	если нет свободных *блоков управления событиями* (`Event Control Block, ECB`).
`OS_PRIO_INVALID`	если заданное значение `prio` больше, чем значение `OS_LOWEST_PRIO`.

Возвращаемое значение – указатель на *блок управления событиями*, если *мьютекс* создан, либо `NULL`, если более нет свободных *блоков управления событиями*.

Замечания 1) *Мьютекс* следует создать перед его использованием.

2) Следует назначать уровень `PIP` более высоким, чем уровень приоритета любой из *Задач*, которые могут пытаться получить доступ к создаваемому *мьютексу*. При этом следует помнить, что не должно быть *Задачи* с уровнем приоритета, равным назначаемому `PIP`.

Пример использования.

```
OS_EVENT *DispMutex;

void main (void) {
  INT8U err;
       ...
  OSInit(); /* Initialize µC/OS-II */
       ...
  DispMutex = OSMutexCreate(20, &err); /* Create Display Mutex */
       ...
  OSStart(); /* Start Multitasking */
}
```

OSMutexDel()

Описание – функция `OSMutexDel()` удаляет ранее созданный *мьютекс*. Использовать эту функцию следует с осторожностью, так как другие *Задачи* могут предполагать существующим удаленный *мьютекс*. Надежный прием состоит в том, чтобы сначала удалить все *задачи*, которые могут использовать подлежащий удалению *мьютекс*.

Файл - os_mutex.c.

Вызов – только из *Задачи*.

Транслируется, если OS_MUTEX_EN && OS_MUTEX_DEL_EN > 0.

Прототип OS_EVENT *OSMutexDel(OS_EVENT *pevent,
INT8U opt,
INT8U *perr);

Аргументы:

`pevent` указатель на *мьютекс*, который ограничивает доступ к ресурсу. Этот указатель возвращает функция `OSMutexCreate()` при создании *мьютекса*.

`opt` определяет, следует ли уничтожить *мьютекс*, только если нет *Задач*, ожидающих его (`opt=OS_DEL_NO_PEND`), или уничтожение требуется вне зависимости от наличия или отсутствия ожидающих *Задач* (`opt=OS_DEL_ALWAYS`). В последнем случае все *Задачи* переводятся *ядром* в состояние готовности к исполнению.

`perr` указатель на переменную, в которую функция возвращает код завершения, одно из следующих значений:

`OS_ERR_NONE`	если вызов функции завершился успешно.
`OS_ERR_DEL_ISR`	если сделана попытка удаления *мьютекса* из обработчика прерывания.
`OS_ERR_INVALID_OPT`	недопустимое значение аргумента `opt`.
`OS_ERR_TASK_WAITING`	если одна или более *задач* ожидают *мьютекса* и задано `opt=OS_DEL_NO_PEND`.
`OS_ERR_EVENT_TYPE`	если аргумент `pevent` указывает не на *мьютекс*.
`OS_ERR_PEVENT_NULL`	если аргумент `pevent` имеет значение `NULL`.

Возвращаемое значение – нулевой указатель, если удаление было успешным, или `pevent`, если удаление не выполнено. В последнем случае следует проверить значение кода завершения.

Замечания 1) Удалять *мьютекс* следует с осторожностью, поскольку некоторые *задачи* могут ожидать (предполагать) существование этого *мьютекса*.

Пример использования.

```
OS_EVENT *DispMutex;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  while (1) {
       ...
    DispMutex = OSMutexDel(DispMutex, OS_DEL_ALWAYS, &err);
    if (DispMutex == (OS_EVENT *)0) {
       /* Mutex has been deleted */
    }
       ...
  }
}
```

OSMutexPend()

Описание – *Задача* вызывает функцию OSMutexPend(), когда ей необходимо получить исключительный доступ к ресурсу. Если ресурс доступен (не занят другой *Задачей*), и *мьютекс* свободен, он переходит в состояние «занят», и функция OSMutexPend() немедленно завершается, возвращая управление вызвавшей функцию *Задаче*. Свидетельством получения *мьютекса* служит сам факт возврата управления и значение кода завершения OS_ERR_NONE. Если мьютекс захвачен другой *Задачей*, то *ядро* (которое получило управление в результате вызова OSMutexPend()), переводит вызвавшую *Задачу* в состояние ожидания и заносит ее в *список ожидания* этого мьютекса. В результате *задача*, вызвавшая OSMutexPend(), ожидает, пока *Задача* владеющая *мьютексом*, не закончит работу с ресурсом и не освободит занятый *мьютекс*, либо пока не истечет заданная величина *тайм-аута*. Если *мьютекс* оказывается освобожден до истечения *тайм-аута, ядро uC/OS* возобновляет выполнение *Задачи* с наивысшим приоритетом из перечня *Задач,* ожидающих этого *мьютекса*. В *uC/OS* реализовано *наследование приоритетов*, это означает, что если *мьютекс* (и ресурс) занят низкоприоритетной *Задачей*, и *задача* с более высоким приоритетом вызывает OSMutexPend(), то *ядро* «поднимает» приоритет *Задачи* удерживающей *мьютекс* до значения PIP, заданного в параметре prio функции OSMutexCreate() при создании этого *мьютекса*.

Файл - os_mutex.c.

Вызов – только из *Задачи*.

Транслируется, если OS_MUTEX_EN > 0.

Прототип void OSMutexPend(OS_EVENT *pevent,
INT16U timeout,
INT8U *perr);

Аргументы:

pevent указатель на *мьютекс,* который ограничивает доступ к ресурсу. Этот указатель возвращает функция OSMutexCreate() при создании *мьютекса*.

timeout задает величину *тайм-аута* в тиках системного таймера, в течение которого *Задача* будет ожидать освобождения *мьютекса*. Значение 0 задает бесконечное время ожидания. Максимальное значение *тайм-аута* составляет 65535 тиков. Начало отсчета *тайм-аута* не синхронизировано с системным таймером, т.е. при величине заданного *тайм-аута* Tm его реальное значение будет находиться между Tm и Tm-1.

perr указатель на переменную, куда будет помещен код завершения, одно из следующих значений:

OS_ERR_NONE если функция завершилась успешно, и *Задача* получила мьютекс.

OS_TIMEOUT если *мьютекс* не освободился в течение *тайм-аута*, заданного параметром timeout.

OS_ERR_EVENT_TYPE если указатель pevent указывает не на *мьютекс*.

OS_ERR_PEVENT_NULL если указатель pevent имеет значение NULL.

OS_ERR_PEND_ISR если сделана попытка захватить *мьютекс* из *обработчика преры вания*.

OS_ERR_PIP_LOWER приоритет *Задачи,* владеющий *мьютексом* выше (т.е. числовое значение приоритета меньше), чем PIP. Это означает, что сделана ошибка при назначении уровня во время создания *мьютекса*: PIP должен быть выше (числовое значение меньше), чем приоритет любой *Задачи*, которая будет использовать этот *мьютекс*. К сожалению, выполнение этого условия невозможно проверить во время создания *мьютекса*, за этим должен следить программист, разрабатывающий приложение.

Возвращаемое значение – отсутствует.

Замечания 1) *Мьютекс* следует создать до его использования.

2) Следует избегать перехода *Задачи*, владеющей *мьютексом* в состояние ожидания (какого-либо другого синхронизирующего объекта: *мьютекса*, *флага* и т.п.). Другими словами, любой ресурс (и соответственно, связанный с ним *мьютекс*) следует использовать и освобождать так быстро, как только это возможно.

Пример использования.

```
OS_EVENT *DispMutex;

void DispTask (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    OSMutexPend(DispMutex, 0, &err);
 /* The only way this task continues is if the mutex is available or signaled! */
       ...
  }
}
```

OSMutexPost()

Описание – вызов функции OSMutexPost() освобождает *мьютекс*. Функцию можно вызывать только, если ранее *Задача* получила *мьютекс* вызовом функций OSMutexAccept() или OSMutexPend(). Если приоритет *Задачи1*, владеющей *мьютексом*, был повышен до PIP в результате запроса *мьютекса Задачей2* с более высоким приоритетом, для *Задачи1* восстанавливается исходный уровень приоритета. Если одна или более *Задач* ожидают *мьютекса*, он отдается наиболее приоритетной *Задаче*. Затем вызывается *планировщик*, чтобы определить, имеет ли *Задача*, получившая мьютекс, наивысший приоритет среди готовых к исполнению, и если это так, выполняется переключение контекста на эту *Задачу*. Если же нет *Задач*, ожидающих *мьютекса*, он просто освобождается.

Файл - os_mutex.c.

Вызов – только из *Задачи*.

Транслируется, если OS_MUTEX_EN > 0.

Прототип INT8U OSMutexPost(OS_EVENT *pevent);

Аргументы:

pevent указатель на *мьютекс*, который ограничивает доступ к ресурсу. Этот указатель возвращает функция OSMutexCreate() при создании *мьютекса*.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	если функция завершилась успешно, и *Задача* освободила *мьютекс*.
OS_ERR_EVENT_TYPE	если указатель pevent указывает не на *мьютекс*.
OS_ERR_PEVENT_NULL	если указатель pevent имеет значение NULL.
OS_ERR_PEND_ISR	если сделана попытка освободить *мьютекс* из *обработчика прерывания*.
OS_ERR_NOT_MUTEX_OWNER	если *задача*, вызвавшая OSMutexPost(OS_EVENT *pevent), не является владельцем *мьютекса* (ошибка программиста).

Замечания 1) *Мьютекс* должен быть создан до его использования.

2) Данную функцию нельзя вызывать из *обработчика прерывания*.

Пример использования.

```
OS_EVENT *DispMutex;
void TaskX (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    err = OSMutexPost(DispMutex);
    switch (err) {
      case OS_ERR_NONE: /* Mutex signaled */
       ...
        break;
      case OS_ERR_EVENT_TYPE:
       ...
        break;
      case OS_ERR_PEVENT_NULL:
       ...
        break;
      case OS_ERR_POST_ISR:
       ...
        break;
    }
       ...
  }
}
```

OSMutexQuery()

Описание – OSMutexQuery() используется, чтобы получить информацию о состоянии *мьютекса*. *задача*, вызывающая функцию , предварительно должна создать структуру данных типа OS_MUTEX_DATA, в которую будут скопированы данные из *блока управления событиями*, связанного с *мьютексом*. Функция позволяет определить, есть ли *задачи*, ожидающие *мьютекса* и каково их количество, путем подсчета «единиц» в поле структуры OSEventTbl[], получить значение , а также определить, свободен или занят *мьютекс*. Размер OSEventTbl[] задается константой OS_EVENT_TBL_SIZE в файле ucos_ii.h.

Файл - os_mutex.c.

Вызов – только из *Задачи*.

Транслируется, если OS_MUTEX_EN && OS_MUTEX_QUERY_EN > 0.

Прототип INT8U OSMutexQuery(OS_EVENT *pevent,
OS_MUTEX_DATA *p_mutex_data);

Аргументы:

pevent указатель на *мьютекс*. Этот указатель возвращает функция OSMutexCreate() при создании *мьютекса*.

p_mutex_data указатель на структуру данных типа OS_MUTEX_DATA, которая содержит следующие поля:

```
INT8U OSMutexPIP; /* The PIP of the mutex */
INT8U OSOwnerPrio; /* The priority of the mutex owner */
INT8U OSValue; /* The current mutex value, 1 means available, */
/* 0 means unavailable */
#if OS_VERSION < 280
INT8U OSEventGrp; /* Copy of the mutex wait list */
INT8U OSEventTbl[OS_EVENT_TBL_SIZE];
#else
INT16U OSEventGrp; /* Copy of the mutex wait list */
INT16U OSEventTbl[OS_EVENT_TBL_SIZE];
#endif
```

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	если вызов функции был успешен.
OS_ERR_EVENT_TYPE	если указатель pevent указывает не на *мьютекс*.
OS_ERR_PEVENT_NULL	если указатель pevent имеет значение NULL.
OS_ERR_PDATA_NULL	если указатель p_mutex_data имеет значение NULL.
OS_ERR_QUERY_ISR	если сделана попытка вызвать OSMutexQuery() из *обработчика прерывания*.

Замечания 1) *Мьютекс* должен быть создан перед его использованием.

2) Недопустимо вызывать функцию OSMutexQuery() из *обработчика прерывания*.

Пример использования.

```
OS_EVENT *DispMutex;

void Task (void *p_arg) {
  OS_MUTEX_DATA mutex_data;
  INT8U err;
  INT8U highest; /* Highest priority task waiting on mutex *
  INT8U x;
  INT8U y;
  for (;;) {
       ...
    err = OSMutexQuery(DispMutex, &mutex_data);
    if (err == OS_ERR_NONE) {
      /* Examine Mutex data */
    }
       ...
  }
}
```

OSQAccept()

Описание – OSQAccept() позволяет определить, имеются ли *сообщения* в заданной *Очереди сообщений*. В отличие от OSQPend(), функция OSQAccept() не переводит вызывающую *задачу* в состояние ожидания, если *сообщения* в *Очереди* отсутствуют. Если *сообщение* имеется, то оно извлекается из *Очереди* и передается вызывающей *Задаче*. Такой вызов часто используется из *обработчика прерывания*, поскольку *обработчик прерывания* не может находиться в состоянии ожидания *сообщения*.

Файл - os_q.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_Q_EN > 0.

Прототип void *OSQAccept(OS_EVENT *pevent,
INT8U *perr);

Аргументы:

pevent указатель на *Очередь сообщений*. Этот указатель возвращает функция OSQCreate() при создании *Очереди*.

perr указатель на переменную, в которую возвращается код ошибки, имеющий одно из следующих значений:

OS_ERR_NONE	если вызов завершился успешно, и в *Очереди* имеется сообщение.
OS_ERR_EVENT_TYPE	если pevent указывает не на *Очередь сообщений*.
OS_ERR_PEVENT_NULL	если pevent имеет значение NULL.
OS_Q_EMPTY	если *Очередь* не содержит сообщений.

Возвращаемое значение – указатель на *сообщение*, если оно доступно, либо NULL, если *Очередь* не содержит *сообщений* или ??? очередное *сообщение* имеет значение NULL. Если *сообщение* доступно, то оно извлекается из *очереди* перед возвратом из OSQAccept().

Замечания 1) *Очередь сообщений* должна быть создана перед ее использованием.

2) Интерфейс этой функции был изменен в версии 2.60, теперь можно помещать в *Очередь* значение NULL. Для этого в функции добавлен аргумент perr.

Пример использования.

```
OS_EVENT *CommQ;

void Task (void *p_arg) {
  void *msg;
  for (;;) {
       ...
    msg = OSQAccept(CommQ); /* Check queue for a message */
    if (msg != (void *)0) {
       /* Message received, process */
       ...
    } else {
       /* Message not received, do something else */
       ...
    }
       ...
  }
}
```

OSQCreate()

Описание – функция создает *Очередь сообщений*, которая затем позволяет *Задаче* или *обработчику прерываний* отправлять *сообщения* длиной в указатель одной или нескольким другим *Задачам*. Смысл *сообщения* определяется спецификой конкретного приложения.

Файл - `os_q.c`.

Вызов – из *Задачи* или из `startup` кода.

Транслируется, если `OS_Q_EN > 0`.

Прототип OS_EVENT *OSQCreate(void **start
INT8U size);

Аргументы:

`start` адрес начала области памяти для *Очереди сообщений*. Эта область памяти должна быть объявлена как массив указателей на `void`.

`size` размер (в элементах) области памяти для *Очереди сообщений*.

Возвращаемое значение – указатель на *блок управления событиями* (*ECB*), связанный с создаваемой *Очередью*. Если нет свободных *ECB*, или нет свободных *блоков управления очередью* QCB, возвращается нулевой указатель.

Замечания 1) *Очередь* должна быть создана до ее использования.

Пример использования.

```
OS_EVENT *CommQ;
void *CommMsg[10];

void main (void) {
  OSInit(); /* Initialize µC/OS-II *
       ...
  CommQ = OSQCreate(&CommMsg[0], 10); // Create COMM Q *
       ...
  OSStart(); /* Start Multitasking *
}
```

OSQDel()

Описание – функция OSQDel() удаляет (уничтожает) ранее созданную *Очередь сообщений*. Пользоваться этой функцией следует аккуратно, так как другие *Задачи* могут предполагать, что уничтоженная *Очередь* существует. Надежный способ состоит в том, чтобы сначала уничтожить все *Задачи*, которые могут обращаться к стираемой *Очереди сообщений*.

Файл - os_q.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_Q_EN > 0.

Прототип OS_EVENT *OSQDel(OS_EVENT *pevent,
INT8U opt,
INT8U *perr);

Аргументы:

pevent указатель на *Очередь сообщений*. Этот указатель возвращает функция OSQCreate() при создании *Очереди*.

opt определяет, следует ли удалять *Очередь сообщений* лишь в случае, когда нет *Задач*, ожидающих этой *очереди* (OS_DEL_NO_PEND), либо удаление следует выполнить безусловно (OS_DEL_ALWAYS). В последнем случае *ядро* переводит все ожидающие *Задачи* в состояние готовности к исполнению.

perr указатель на переменную, в которую помещается один из следующих кодов завершения:

OS_ERR_NONE	вызов функции завершился успешно, *Очередь* сообщений удалена.
OS_ERR_DEL_ISR	попытка удалить *Очередь сообщений* из обработчика прерывания.
OS_ERR_INVALID_OPT	неверное значение аргумента opt.
OS_ERR_TASK_WAITING	если одна или более *задач* ожидают сообщений из *Очереди*.
OS_ERR_EVENT_TYPE	если pevent указывает НЕ на *Очередь сообщений*.
OS_ERR_PEVENT_NULL	если pevent имеет значение NULL.

Возвращаемое значение – нулевой указатель, если *Очередь* успешно удалена, либо pevent, если удаления не произошло. В последнем случае следует анализировать значение кода завершения *perr для выяснения причины.

Замечания 1) Эту функцию следует использовать аккуратно, поскольку другие *Задачи* могут предполагать существование уничтоженных *Задач*.

2) Прерывания запрещены в течение времени, которое требуется *ядру*, чтобы сделать готовыми *Задачи*, ожидающие *сообщений* из уничтоженной *Очереди*. Следовательно, задержка реакции на запрос прерывания, пришедшего во время уничтожения *Очереди*, зависит от количества *Задач*, ожидавших этой *Очереди*.

Пример использования.

```
OS_EVENT *DispQ;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  while (1) {
       ...
    DispQ = OSQDel(DispQ, OS_DEL_ALWAYS, &err);
    if (DispQ == (OS_EVENT *)0) {
       /* Queue has been deleted */
    }
       ...
  }
}
```

OSQFlush()

Описание – функция очищает содержимое *Очереди сообщений*. Время выполнения этой функции не зависит от количества сообщений в очереди и от количества *Задач*, ожидающих этой *Очереди*.

Файл - `os_q.c`.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_Q_EN && OS_Q_FLUSH_EN > 0.

Прототип INT8U *OSQFlush(OS_EVENT *pevent);

Аргументы:

`pevent` указатель на *Очередь сообщений*. Этот указатель возвращает функция `OSQCreate()` при создании *Очереди*.

Возвращаемое значение – код завершения, который может принимать одно из следующих значений:

`OS_ERR_NONE`	вызов функции завершился успешно, *Очередь* сообщений удалена.
`OS_ERR_EVENT_TYPE`	если `pevent` указывает НЕ на *Очередь сообщений*.
OS_ERR_PEVENT_NULL	если pevent имеет значение NULL.

Замечания 1) *Очередь сообщений* должна быть создана до ее использования.

2) Эту функцию следует использовать аккуратно, поскольку очистка *Очереди* приведет к потере хранившихся в ней указателей, и следовательно к потере информации о расположении в памяти объектов, на которые ссылались удаленные указатели. Если память под эти объекты выделялась динамически, то ее освобождение может стать невозможным, т.е. может возникнуть «утечка памяти».

Пример использования.

```
OS_EVENT *CommQ;

void main (void) {
  INT8U err;
  OSInit(); /* Initialize µC/OS-II */
       ...
  err = OSQFlush(CommQ);
       ...
  OSStart(); /* Start Multitasking */
}
```

OSQPend()

Описание – OSQPend() используется, когда *Задаче* необходимо получить *сообщение* из *Очереди сообщений*. *Сообщения* могут отсылаться данной *Задаче* либо из других *Задач*, либо из *обработчиков прерываний*. *Сообщение* представляет собой переменную размером с указатель, смысл сообщения определяется программистом, исходя из конкретной задачи. Если в *очереди* имеется хотя бы одно *сообщение*, функция OSQPend() немедленно завершается, и *сообщение* передается вызвавшей *Задаче*. Если *Очередь сообщений* пуста, вызвавшая *Задача* переводится *ядром* в состояние ожидания до тех пор, пока в *Очереди* не появится *сообщение*, либо пока не истечет заданный программистом *тайм-аут*. Если несколько *Задач* ожидают сообщения из *Очереди*, и оно появилось, будет возобновлена *Задача* с наивысшим приоритетом (из ожидающих). Следует помнить, что если *Задача* ожидающая сообщения, была вдобавок переведена другой *Задачей* в состояние ожидания посредством функции OSTaskSuspend(), то ожидающая *Задача* получит сообщение, но будет переведена в состояние готовности только после вызова функции OSTaskResume() (какой-либо другой *Задачей*).

Файл - os_q.c.

Вызов – только из *Задачи*.

Транслируется, если OS_Q_EN > 0.

Прототип
```
void *OSQPend(OS_EVENT *pevent,
              INT16U timeout,
              INT8U *perr);
```

Аргументы:

pevent указатель на *Очередь сообщений*. Этот указатель возвращает функция OSQCreate() при создании *Очереди*.

timeout ненулевое значение этого параметра приводит к возобновлению выполнения *Задачи*, ожидавшей сообщения из *Очереди*, если оно не появилось в течение заданного параметром количества тиков системного таймера. Максимальное значение *тайм-аута* равно 65535. Нулевое значение параметра вызывает бесконечное ожидание сообщения. Начало *тайм-аута* не синхронизировано с работой системного таймера, т.е. при заданном значении Tm реальное значение *тайм-аута* составит величину между Tm и Tm-1.

perr указатель на переменную, в которую помещается один из следующих кодов завершения:

OS_ERR_NONE вызов функции завершился успешно, *Очередь* сообщений удалена.

OS_ERR_TIMEOUT если сообщение не получено в течение заданного *тайм-аута*.

OS_ERR_TASK_WAITING если одна или более *задач* ожидают сообщений из *очереди*.

OS_ERR_EVENT_TYPE если pevent указывает не на *очередь сообщений*.

OS_ERR_PEVENT_NULL если pevent имеет значение NULL.

OS_ERR_PEND_ISR если сделана попытка вызова функции из *обработчика прерывания*. Это недопустимо, и *ядро* проверяет эту ситуацию. *Обработчик прерывания* при этом (в отличие от *Задачи*) не будет приостановлен.

Возвращаемое значение – сообщение (переменная размером с указатель), извлеченное из *Очереди*, при этом значение кода завершения равно OS_ERR_NONE. Если истек *тайм-аут*, то возвращаемое значение равно NULL, а код завершения равен OS_TIMEOUT.

Замечания 1) *Очередь сообщений* должна быть создана до ее использования.

2) Недопустимо вызывать функцию OSQPend() из *обработчика прерываний*.

3) Начиная с версии 2.60 допустимо помещать в *Очередь сообщений* значение NULL в качестве *сообщения*.

Пример использования.

```
OS_EVENT *CommQ;

void CommTask(void *p_arg) {
  INT8U err;
  void *msg;
  for (;;) {
       ...
    msg = OSQPend(CommQ, 100, &err);
    if (err == OS_ERR_NONE) {
       /* Message received within 100 ticks! */
       ...
    } else {
       /* Message not received, must have timed out */
       ...
    }
       ...
  }
}
```

OSQPendAbort()

Новая функция, появилась в вер.2.84

Описание – функция принудительно переводит в состояние готовности *Задачи*, ожидающие сообщения из данной *Очереди сообщений*. Эту функцию следует использовать для аварийного возврата ожидающих *Задач* в состояние готовности, как альтернативу нормальному использованию функций `OSQPost()`, `OSQPostFront()` или `OSQPostOpt()`.

Файл - `os_q.c`.

Вызов – только из *Задачи*.

Транслируется, если (OS_Q_EN > 0)&(OS_Q_PEND_ABORT_EN>0).

Прототип
```
void *OSQPendAbort(OS_EVENT *pevent,
                   INT16U   timeout,
                   INT8U    *perr);
```

Аргументы:

`pevent` указатель на *Очередь сообщений*. Этот указатель возвращает функция `OSQCreate()` при создании *Очереди*.

`opt` тип аварийного завершения:

`OS_PEND_OPT_NONE` - аварийно переводит в состояние готовности лишь одну *Задачу* с наивысшим приоритетом из ожидающих указанной *Очереди Сообщений*;

`OS_PEND_OPT_BROADCAST` – переводит в состояние готовности все *Задачи*, ожидающие указанной *Очереди Сообщений*.

perr указатель на переменную, в которую функция вернет код завершения, одно из следующих значений:

OS_ERR_NONE если нет *Задач*, ожидающих указанного *Очереди*;

OS_ERR_PEND_ABORT хотя бы одна *Задача*, ожидавшая *Очереди*, была переведе на в состояние готовности, возвращаемое значение содержит число таких *Задач*;

OS_ERR_EVENT_TYPE если pevent указывает не на *Очередь Сообщений*;

OS_ERR_PEVENT_NULL если `pevent` имеет значение NULL.

Возвращаемое значение – число *Задач,* переведенных в состояние готовности. Нуль указывает, что таких *Задач* не было.

Замечания 1) *Очередь сообщений* должна быть создана до ее использования.

Пример использования.

```
OS_EVENT *CommQ;

void CommTask(void *p_arg) {
  INT8U err;
  INT8U nbr_tasks;
  (void)p_arg;
  for (;;) {

    nbr_tasks = OSQPendAbort(CommQ, OS_PEND_OPT_BROADCAST, &err);
    if (err == OS_ERR_NONE) {
       /* No tasks were waiting on the queue */
    } else {
       /* All pends of tasks waiting on queue were aborted    */
       /* ... 'nbr_tasks' indicates how many were made ready. */
    }
...
  }
}
```

OSQPost()

Описание – OSQPost() посылает *сообщение* в *Очередь*. *Сообщение* представляет собой переменную размером в указатель, ее смысл определяется программистом, исходя из решаемой задачи. Если *Очередь сообщений* полна, функция немедленно завершается, сообщение не помещается в очередь, а вызывающей *Задаче* возвращается код ошибки OS_Q_FULL. Если какие-либо *Задачи* ожидают сообщений из *Очереди*, пришедшее сообщение передается *Задаче* с наивысшим приоритетом (среди ожидающих сообщения из этой *Очереди*). Если приоритет ожидающей *Задачи* выше приоритета *Задачи*, отправившей *сообщение*, то *Задача*-отправитель приостанавливается, а *Задача*-получатель возобновляет выполнение. *Очередь сообщений* реализует при использовании функции OSQPost() алгоритм FIFO, т.е. раньше отправленное *сообщение* раньше извлекается.

Файл - os_q.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_Q_EN && OS_Q_POST_EN > 0.

Прототип INT8U OSQPost(OS_EVENT *pevent,
void *msg);

Аргументы:

pevent указатель на *Очередь сообщений*. Этот указатель возвращает функция OSQCreate() при создании *Очереди*.

msg передаваемое *сообщение*, переменная размером с указатель. Начиная с версии 2.60, разрешается передавать в качестве сообщения NULL.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	вызов функции завершился успешно, сообщение помещено в *Очередь*.
OS_Q_FULL	в *Очереди* нет свободного места.
OS_ERR_EVENT_TYPE	pevent указывает НЕ на *Очередь сообщений*.
OS_ERR_PEVENT_NULL	pevent имеет значение NULL.

Замечания 1) *Очередь сообщений* должна быть создана до ее использования.
2) Начиная с версии 2.60, можно посылать в *очередь сообщений* значение NULL.

Пример использования.

```
OS_EVENT *CommQ;
INT8U CommRxBuf[100];
void CommTaskRx (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    err = OSQPost(CommQ, (void *)&CommRxBuf[0]);
    switch (err) {
      case OS_ERR_NONE:    /* Message was deposited into queue */
       ...
        break;
      case OS_Q_FULL:      /* Queue is full */
       ...
        break;
       ...
    }
       ...
  }
}
```

OSQPostFront()

Описание – OSQPostFront() посылает сообщение в начало очереди, т.е. функция OSQPostFront() реализует порядок last-in first-out (LIFO), характерный для стекового доступа. Сообщение представляет собой переменную размером в указатель, ее смысл определяется программистом, исходя из решаемой задачи. Если *Очередь сообщений* полна, функция немедленно завершается, сообщение не помещается в *Очередь*, а вызывающей *Задаче* возвращается код ошибки OS_Q_FULL. Если какие-либо *Задачи* ожидают сообщений из *Очереди*, пришедшее сообщение передается *Задаче* с наивысшим приоритетом (среди ожидающих сообщения из этой *Очереди*). Если приоритет ожидающей *Задачи* выше приоритета *Задачи*, отправившей *сообщение*, то *Задача*-отправитель приостанавливается, а *Задача*-получатель возобновляет выполнение.

Файл - os_q.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_Q_EN && OS_Q_POST_FRONT_EN > 0.

Прототип
```
INT8U OSQPostFront(OS_EVENT *pevent,
                   void *msg);
```

Аргументы:

pevent	указатель на *Очередь сообщений*. Этот указатель возвращает функция OSQCreate() при создании *Очереди*.
msg	передаваемое *сообщение*, переменная размером с указатель. Начиная с версии 2.60, разрешается передавать в качестве сообщения NULL.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	вызов функции завершился успешно, сообщение помещено в *Очередь*.
OS_Q_FULL	в *Очереди* нет свободного места.
OS_ERR_EVENT_TYPE	pevent указывает НЕ на *Очередь сообщений*.
OS_ERR_PEVENT_NULL	pevent имеет значение NULL.

Замечания 1) *Очередь сообщений* должна быть создана до ее использования.
2) Начиная с версии 2.60, можно посылать в *очередь сообщений* значение NULL.

Пример использования.

```
OS_EVENT *CommQ;
INT8U CommRxBuf[100];

void CommTaskRx (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    err = OSQPostFront(CommQ, (void *)&CommRxBuf[0]);
    switch (err) {
      case OS_ERR_NONE:
       /* Message was deposited into queue */
       ...
        break;
      case OS_Q_FULL:
       /* Queue is full */
       ...
      break;
       ...
    }
       ...
  }
}
```

OSQPostOpt()

Описание – OSQPostOpt() посылает сообщение в *Очередь*. Сообщение представляет собой переменную размером в указатель, ее смысл определяется программистом, исходя из решаемой задачи. Если *Очередь сообщений* полна, функция немедленно завершается, сообщение не помещается в *Очередь*, а вызывающей *Задаче* возвращается код ошибки OS_Q_FULL. Если какие-либо *Задачи* ожидают сообщений из *Очереди*, функция OSQPostOpt() позволяет либо отправить сообщение только *Задаче* с наивысшим приоритетом (opt = OS_POST_OPT_NONE), либо всем *Задачам*, ожидающим сообщений из этой *Очереди* (opt = OS_POST_OPT_BROADCAST). В любом из этих случаев *ядро* выполняет *планирование* – если среди *Задач*, получивших *сообщение*, есть *Задача* с приоритетом, более высоким, чем у *Задачи*-отправителя, то отправитель переводится в состояние *ожидания*, и выполняется переключение на высокоприоритетную *Задачу*-получатель.

OSQPostOpt() эмулирует поведение функций OSQPost() и OSQPostFront() а, кроме того, позволяет осуществлять «широковещательную» передачу. Это означает, что, если предполагается использовать функцию , то можно отключить компиляцию функций OSQPost() и OSQPostFront() для уменьшения объема кода.

Файл - os_q.c.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_Q_EN && OS_Q_POST_OPT_EN P > 0.

Прототип INT8U OSQPostOpt(OS_EVENT *pevent,
void *msg,
INT8U opt);

Аргументы:

pevent указатель на *Очередь сообщений*. Этот указатель возвращает функция OSQCreate() при создании *Очереди*.

msg передаваемое *сообщение*, переменная размером с указатель. Начиная с версии 2.60, разрешается передавать в качестве сообщения NULL.

opt определяет тип операции POST:

OS_POST_OPT_NONE посылка одной *Задаче* (эмуляция OSQPost().
OS_POST_OPT_FRONT эмуляция поведения функции OSQPostFront().
OS_POST_OPT_BROADCAST посылка всем *Задачам*, ожидающим данной *Очереди*.
OS_POST_OPT_FRONT + OS_POST_OPT_BROADCAST эмуляция OSQPostFront() с широковещательной передачей.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE вызов функции завершился успешно, сообщение помещено в *Очередь*.

OS_Q_FULL в *Очереди* нет свободного места.

OS_ERR_EVENT_TYPE pevent указывает НЕ на *Очередь сообщений*.

OS_ERR_PEVENT_NULL pevent имеет значение NULL.

Замечания 1) *Очередь сообщений* должна быть создана до ее использования.

2) Если необходимо использовать эту функцию и одновременно требуется минимизировать объем кода, можно запретить компиляцию функций OSQPost() (установив OS_Q_POST_EN в 0) и OSQPostFront() (установив OS_Q_POST_FRONT_EN в 0 в файле OS_CFG.H).

3) Время выполнения функции OSQPostOpt() зависит от количества *Задач*, ожидающих *сообщений* из *Очереди*, если выбрана широковещательная передача.

Пример использования.

```
OS_EVENT *CommQ;
INT8U CommRxBuf[100];

void CommRxTask (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    err = OSQPostOpt(      CommQ,
                           (void *)&CommRxBuf[0],
                           OS_POST_OPT_BROADCAST);
       ...
  }
}
```

OSQQuery()

Описание – функция `OSQQuery()` позволяет получить информацию о состоянии *Очереди сообщений*. Пользовательское приложение должно объявить структуру данных типа `OS_Q_DATA`, в которую функция скопирует данные из *блока управления событием*, связанного с данной *Очередью сообщений*. Функция `OSQQuery()` позволяет узнать, есть ли *Задачи*, ожидающие сообщений из *Очереди*, и сколько таких *Задач* (для этого следует подсчитать количество «единиц» в поле `OSEventTbl[]` структуры `OS_Q_DATA`, сколько сообщений имеется в *Очереди*, и каков размер *Очереди*. Функция `OSQQuery()` позволяет также получить «следующее» сообщение, которое будет возвращено, если *Очередь* не пуста. Следует помнить, что размер поля `OSEventTbl[]` задается константой `OS_EVENT_TBL_SIZE` в файле `ucos_ii.h`.

Файл - `os_q.c`.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_Q_EN && OS_QUERY_EN > 0.

Прототип INT8U OSQQuery(OS_EVENT *pevent,
OS_Q_DATA *p_q_data);

Аргументы:

`pevent` указатель на *очередь сообщений*. Этот указатель возвращает функция `OSQCreate()` при создании *очереди*.

`p_q_data` указатель на структуру данных типа `OS_Q_DATA`, которая содержит следующие поля:

```
void *OSMsg; /* Следующее сообщение, если оно есть */
INT16U OSNMsgs; /* Количество сообщений в очереди */
INT16U OSQSize; /* Size of the message queue */
#if OS_VERSION < 280
INT8U OSEventTbl[OS_EVENT_TBL_SIZE]; /* Лист ожидания очереди сообщений */
INT8U OSEventGrp;
#else
INT16U OSEventTbl[OS_EVENT_TBL_SIZE]; /* Лист ожидания очереди сообщений */
INT16U OSEventGrp;
#endif
```

Возвращаемое значение – код завершения, одно из следующих значений:

`OS_ERR_NONE` вызов функции завершился успешно, сообщение помещено в *очередь*.
`OS_ERR_EVENT_TYPE` `pevent` указывает НЕ на *Очередь сообщений*.
`OS_ERR_PEVENT_NULL` `pevent` имеет значение `NULL`.
`OS_ERR_PDATA_NULL` `p_q_data` имеет значение `NULL`.

Замечания 1) *Очередь сообщений* должна быть создана перед ее использованием.

Пример использования.

```
OS_EVENT *CommQ;

void Task (void *p_arg) {
  OS_Q_DATA qdata;
  INT8U err;
  for (;;) {
       ...
    err = OSQQuery(CommQ, &qdata);
    if (err == OS_ERR_NONE) {           /* 'qdata' can be examined! */
    }
  }
}
```

OSSchedLock()

Описание – OSSchedLock() запрещает *ядру* выполнять *диспетчеризацию* (переключение *Задач*), до тех пор, пока не будет выполнена функция OSSchedUnlock(). Функция, вызвавшая OSSchedLock(), выполняется (и не вытесняется), даже если становятся готовыми к исполнению *Задачи* с более высоким приоритетом. Однако, запросы прерываний вызывают переключение на обработчики прерываний (если разрешены аппаратные прерывания процессору). Функции OSSchedLock() и OSSchedUnlock() должны обязательно использоваться «в паре». Система uC/OS-II позволяет вложенные вызовы OSSchedLock() на глубину до 255. Планирование и диспетчеризация будут возобновлены после того, как будет выполнено соответствующее количество вызовов функции OSSchedUnlock().

Файл - os_core.c.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_SCHED_LOCK_EN>0.

Прототип void OSSchedLock(void);

Аргументы: нет.

Возвращаемое значение – нет.

Замечания 1) После вызова OSSchedLock() приложение не должно вызывать системных сервисов, которые могут перевести текущую *Задачу* в состояние ожидания, т.е. вызовы функций OSTimeDly(), OSTimeDlyHMSM(), OSFlagPend(), OSSemPend(), OSMutexPend(), OSMboxPend(), или OSQPend(). Поскольку *Диспетчер Задач* заблокирован, никакая другая *Задача*, кроме текущей, не сможет выполняться, и система «повиснет».

Пример использования.

```
void TaskX (void *p_arg) {
  for (;;) {
       ...
    OSSchedLock(); /* Prevent other tasks to run */
       ...
       /* Code protected from context switch */
       ...
    OSSchedUnlock(); /* Enable other tasks to run */
       ...
  }
}
```

OSSchedUnlock()

Описание – функция OSSchedUnlock() возобновляет *планирование* и *диспетчеризацию*, если она используется «в паре» с функцией OSSchedLock().

Файл - os_core.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_SCHED_LOCK_EN>0.

Прототип void OSSchedUnlock(void);

Аргументы: нет.

Возвращаемое значение – нет.

Замечания 1) После вызова OSSchedLock() приложение не должно вызывать системных сервисов, которые могут перевести текущую *задачу* в состояние ожидания, т.е. вызовы функций OSTimeDly(), OSTimeDlyHMSM(), OSFlagPend(), OSSemPend(), OSMutexPend(), OSMboxPend(), или OSQPend(). Поскольку *Диспетчер Задач* заблокирован, никакая другая *Задача*, кроме текущей, не сможет выполняться, и система «повиснет».

Пример использования.

```
void TaskX (void *p_arg) {
  for (;;) {
       ...
    OSSchedLock(); /* Prevent other tasks to run */
       ...
       /* Code protected from context switch */
       ...
    OSSchedUnlock(); /* Enable other tasks to run */
       ...
  }
}
```

OSSemAccept()

Описание – функция `OSSemAccept()` позволяет проверить, доступен ли *ресурс* (или произошло ли *событие*). В отличие от функции `OSSemPend()`, данная функция не переводит вызвавшую *Задачу* в состояние *ожидания*, если *ресурс* занят. Допускается использование функции `OSSemAccept()` для овладения *семафором* из *обработчика прерывания*.

Файл - `os_sem.c`.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_SEM_EN && OS_SEM_ACCEPT_EN > 0.

Прототип `INT16U OSSemAccept(OS_EVENT *pevent);`

Аргументы:

`pevent` указатель на *Семафор*, охраняющий *ресурс*. Этот указатель возвращает функция `OSSemCreate()` при создании *Семафора*.

Возвращаемое значение – Если при вызове функции `OSSemAccept()` значение *Семафора* больше нуля, то это значение декрементируется, функция `OSSemAccept()` завершается, а вызвавшей *Задаче* возвращается значение *Семафора* ДО декремента (и следовательно, оно больше нуля). Если значение *Семафора* при вызове функции `OSSemAccept()` уже равно нулю, это означает, что ресурс недоступен, функция завершается, а вызвавшей *Задаче* возвращается нуль.

Замечания 1) *Семафор* должен быть создан до его использования.

Пример использования.

```
OS_EVENT *DispSem;

void Task (void *p_arg) {
  INT16U value;
  for (;;) {
    value = OSSemAccept(DispSem); /* Check resource availability */
    if (value > 0) {
       /* Resource available, process */
       ...
    }
       ...
  }
}
```

OSSemCreate()

Описание – функция `OSSemCreate()` создает и инициализирует *семафор*. Он позволяет *Задаче* синхронизировать свои действия с другой *Задачей* или с *обработчиком прерывания* (для этого *семафор* нужно инициализировать в 0). Другое назначение *семафора* – обеспечивать исключительный доступ к экземплярам множественного *ресурса*, для этого *семафор* надо проинициализировать значением, равным количеству экземпляров ресурса.

Файл - `os_sem.c`.

Вызов – из *Задачи* или из `startup` кода.

Транслируется, если `OS_SEM_EN > 0`.

Прототип OS_EVENT *OSSemCreate(INT16U value);

Аргументы:

`value` начальное значение *семафора*, которое может находиться в диапазоне от 0 до 65536. Значение 0 используется для обозначения того, что *ресурс* недоступен, либо что событие не произошло.

Возвращаемое значение – указатель на *блок управления событиями*, связанный с данным *семафором*. Если не свободных *блоков управления событиями*, функция возвращает нулевой указатель.

Замечания 1) Семафор должен быть создан перед его использованием.

Пример использования.

```
OS_EVENT *DispSem;

void main (void) {
       ...
  OSInit(); /* Initialize µC/OS-II */
       ...
  DispSem = OSSemCreate(1); /* Create Display Semaphore */
       ...
  OSStart(); /* Start Multitasking */
}
```

OSSemDel()

Описание – функция используется для уничтожения *семафора*. Использовать эту функцию следует с осторожностью, поскольку другие *Задачи* могут пытаться получить доступ к удаленному *семафору*. Хорошая практика – удалить все *Задачи*, которые могут использовать данный *семафор* ДО его удаления.

Файл - `os_sem.c`.

Вызов – только из *Задачи*.

Транслируется, если OS_SEM_EN && OS_SEM_DEL_EN > 0.

Прототип OS_EVENT *OSSemDel(OS_EVENT *pevent,
INT8U opt,
INT8U *perr);

Аргументы:

`pevent` указатель на *семафор*. Этот указатель возвращает функция `OSSemCreate()` при создании *семафора*.

`opt` определяет, следует ли удалять *семафор* лишь в случае, когда нет *Задач*, ожидающих этого *семафора* `(OS_DEL_NO_PEND)`, либо удаление следует выполнить безусловно `(OS_DEL_ALWAYS)`. В последнем случае *Ядро* переводит все ожидающие *Задачи* в состояние готовности к исполнению.

`perr` указатель на переменную, в которую помещается один из следующих кодов завершения:

`OS_ERR_NONE`	вызов функции завершился успешно, *семафор* удален.
`OS_ERR_DEL_ISR`	попытка удалить *семафор* из обработчика прерывания.
`OS_ERR_INVALID_OPT`	неверное значение аргумента `opt`.
`OS_ERR_TASK_WAITING`	если одна или более *Задач* ожидают *семафора*.
`OS_ERR_EVENT_TYPE`	если `pevent` указывает не на *семафор*.
`OS_ERR_PEVENT_NULL`	если `pevent` имеет значение `NULL`.

Возвращаемое значение – `NULL`, если удаление *семафора* было успешным, либо `pevent`, если удаления не произошло. В последнем случае следует проверять код завершения для выяснения причины ошибки.

Замечания 1) Функцию удаления следует применять аккуратно, поскольку другие *Задачи* могут предполагать наличие удаленного *семафора*.

2) Прерывания запрещены на время, в течение которого *ядро* переводит *Задачи,* ожидающие *семафора* в состояние готовности к исполнению, иными словами, задержка реакции на запрос прерывания может зависеть от количества *Задач*, ожидающих *семафора*.

Пример использования.

```
OS_EVENT *DispSem;

void Task (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    DispSem = OSSemDel(DispSem, OS_DEL_ALWAYS, &err);
    if (DispSem == (OS_EVENT *)0) {
       /* Semaphore has been deleted */
       ...
    }
       ...
  }
}
```

OSSemPend()

Описание – OSSemPend() используется, если *Задача* хочет получить исключительный доступ к *ресурсу*, либо синхронизировать свои действия с другой *Задачей* или *обработчиком прерывания*, либо ожидает наступления некоторого события. Если *Задача* вызывает функцию OSSemPend() и при этом значение *семафора* больше нуля, функция декрементирует значение семафора, и возвращает управление вызвавшей *Задаче*. Если же значение *семафора* равно нулю, *ядро* переводит вызвавшую *Задачу* в состояние ожидания, и заносит ее в список ожидания данного *семафора*. Приостановленная *Задача* будет ожидать либо пока другая *Задача* или *обработчик прерывания* не пошлет сигнал (не вызовет функцию OSSemPost()) для данного *семафора*, либо пока не истечет тайм-аут (если он задан). Если *семафор* вышел из «нулевого» состояния до истечения тайм-аута, *ядро* возобновляет выполнение *Задачи* с наивысшим приоритетом, из *Задач*, ожидающих данного *семафора*. При этом *Задача*, ожидавшая данного *семафора*, но приостановленная вызовом функции OSTaskSuspend(), «получит» семафор (т.е. будет исключена из его списка ожидания). Но при этом данная *Задача* не перейдет в состояние готовности, пока не будет (другой *Задачей*) вызвана OSTaskResume().

Файл - os_sem.c.

Вызов – только из *Задачи*.

Транслируется, если OS_SEM_EN > 0.

Прототип void OSSemPend(OS_EVENT *pevent,
INT16U timeout,
INT8U *perr);

Аргументы:

pevent указатель на *семафор*. Этот указатель возвращает функция OSSemCreate() при создании *семафора*.

timeout задает величину *тайм-аута* в тиках системного таймера, в течение которого *Задача* будет ожидать *семафора*. Значение 0 задает бесконечное время ожидания. Максимальное значение *тайм-аута* составляет 65535 тиков. Начало отсчета *тайм-аута* не синхронизировано с системным таймером, т.е. при величине заданного *тайм-аута* Tm его реальное значение будет находиться между Tm и Tm-1.

perr указатель на переменную, в которую будет помещен код завершения, одно из следующих значений:

OS_ERR_NONE если функция завершилась успешно, и *Задача* получила *семафор*.
OS_TIMEOUT если *семафор* не получил сигнала в течение *тайм-аута*, заданного параметром timeout.
OS_ERR_EVENT_TYPE если указатель pevent указывает НЕ на *семафор*.
OS_ERR_PEVENT_NULL если указатель pevent имеет значение NULL.
OS_ERR_PEND_ISR если сделана попытка получить *семафор* из *обработчика прерывания*.

Возвращаемое значение – нет.

Замечания 1) *Семафор* должен быть создан перед его использованием.

Пример использования.

```
OS_EVENT *DispSem;

void DispTask (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    OSSemPend(DispSem, 0, &err);
/* Задача выйдет из ожидания только после освобождения семафора!!! */
       ...
  }
}
```

OSSemPendAbort()

Новая функция, появилась в вер.2.84

Описание – функция принудительно переводит в состояние готовности *Задачи*, ожидающие указанного *Семафора*. Эту функцию следует использовать для аварийного возврата ожидающих *Задач* в состояние готовности, как альтернативу нормальному использованию функции `OSSemPost()`.

Файл - `os_sem.c`.

Вызов – только из *Задачи*.

Транслируется, если (`OS_ SEM _EN > 0)&( OS_ SEM _PEND_ABORT_EN>0)`.

Прототип
```
INT8U OSSemPendAbort(OS_EVENT *pevent,
                     INT16U   timeout,
                     INT8U    *perr);
```

Аргументы:

`pevent` указатель на *семафор*. Этот указатель возвращает функция `OSSemCreate()` при создании *семафора*.

`opt` тип аварийного завершения:

`OS_PEND_OPT_NONE` - аварийно переводит в состояние готовности лишь одну *Задачу* с наивысшим приоритетом из ожидающих указанного *семафора*;

`OS_PEND_OPT_BROADCAST` – переводит в состояние готовности все *Задачи*, ожидающие указанного *семафора*.

`perr` указатель на переменную, в которую функция вернет код завершения, одно из следующих значений:

`OS_ERR_NONE` если нет *Задач*, ожидающих указанного *семафора*;

`OS_ERR_PEND_ABORT` хотя бы одна *Задача*, ожидавшая *семафора*, была переведена в состояние готовности; возвращаемое значение содержит число таких *Задач*;

`OS_ERR_EVENT_TYPE` если `pevent` указывает не на *семафор*;

`OS_ERR_PEVENT_NULL` если `pevent` имеет значение `NULL`.

Возвращаемое значение – число *Задач,* переведенных в состояние готовности. Нуль указывает, что таких *Задач* не было.

Замечания 1) *Семафор* должен быть создан до его использования.

Пример использования.

```
OS_EVENT *CommSem;
void CommTask(void *p_arg) {
  INT8U err;
  INT8U nbr_tasks;
  (void)p_arg;
  for (;;) {

    nbr_tasks = OSSemPendAbort(CommSem, OS_PEND_OPT_BROADCAST, &err);
    if (err == OS_ERR_NONE) {
      /* No tasks were waiting on the semaphore */
    } else {
      /* All pends of tasks waiting on semaphore were aborted. */
      /* ... 'nbr_tasks' indicates how many were made ready.   */
...
    }
  }
}
```

OSSemPost()

Описание – *семафор* получает «сигнал» в результате вызова функции `OSSemPost()`, произведенного какой-то другой *Задачей* или *обработчиком прерывания*. При этом значение *семафора* инкрементируется, и управление возвращается *Задаче* вызвавшей `OSSemPost()`. Если какие-либо *Задачи* ожидают этого *семафора*, *ядро* переводит в состояние готовности ту из них, которая имеет наивысший приоритет. Если этот приоритет выше, чем у *Задачи*, вызвавшей `OSSemPost()`, происходит переключение контекста, и ожидавшая *семафора* *Задача* возобновляет исполнение.

Файл - `os_sem.c`.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если `OS_SEM_EN > 0`.

Прототип INT8U OSSemPost(OS_EVENT *pevent);

Аргументы:

`pevent` указатель на *семафор*. Этот указатель возвращает функция `OSSemCreate()` при создании *семафора*.

Возвращаемое значение – код завершения, одно из следующих значений:

`OS_ERR_NONE`	если *семафор* успешно получил «сигнал», и его значение инкрементировано.
`OS_SEM_OVF`	если значение *семафора* переполнилось при инкременте.
`OS_ERR_EVENT_TYPE`	если указатель `pevent` указывает не на *семафор*.
`OS_ERR_PEVENT_NULL`	если указатель `pevent` имеет значение `NULL`.

Замечания 1) *Семафор* должен быть создан до его использования.

Пример использования.

```
OS_EVENT *DispSem;
  void TaskX (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    err = OSSemPost(DispSem);
    switch (err) {
      case OS_ERR_NONE:          /* семафор инкрементирован */
       ...
        break;
      case OS_SEM_OVF:           /* семафор переполнился */
       ...
        break;
       ...
    }
       ...
  }
}
```

OSSemQuery()

Описание – функция OSSemQuery() позволяет получить информацию о состоянии *семафора*. В программе следует объявить структуру типа OS_SEM_DATA, в которую функция OSSemQuery() скопирует данные из *блока управления событиями*, связанного с данным *семафором*, а именно: сколько *задач* ожидают данного *семафора* (для этого следует подсчитать количество «единиц» в поле OSEventTbl[] структуры), и каково значение *семафора*. Размер поля OSEventTbl[]в структуре типа OS_SEM_DATA задается константой OS_EVENT_TBL_SIZE в файле ucos_ii.h.

Файл - os_sem.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_SEM_EN && OS_SEM_QUERY_EN > 0.

Прототип INT8U OSSemQuery(OS_EVENT *pevent,
OS_SEM_DATA *p_sem_data);

Аргументы:

pevent указатель на *семафор*. Этот указатель возвращает функция OSSemCreate() при создании *семафора*.

P_sem_data указатель на структуру типа OS_SEM_DATA, которая содержит следующие поля:

```
INT16U OSCnt;       /* Текущее значение счетчика семафора */
#if OS_VERSION < 280
INT8U OSEventTbl[OS_EVENT_TBL_SIZE]; /* Список ожидания семафора */
INT8U OSEventGrp;
#else
INT16U OSEventTbl[OS_EVENT_TBL_SIZE]; /* Список ожидания семафора */
INT16U OSEventGrp;
#endif
```

Возвращаемое значение – код завершения, один из следующих:

OS_ERR_NONE	если вызов завершился успешно.
OS_ERR_EVENT_TYPE	если указатель pevent указывает не на *семафор*.
OS_ERR_PEVENT_NULL	если указатель pevent имеет значение NULL.
OS_ERR_PDATA_NULL	если указатель p_sem_data имеет значение NULL.

Замечания 1) *Семафор* должен быть создан до его использования.

Пример использования.

```
OS_EVENT *DispSem;
void Task (void *p_arg) {
  OS_SEM_DATA sem_data;
  INT8U err;
  INT8U x;
  INT8U y;
  for (;;) {
       ...
    err = OSSemQuery(DispSem, &sem_data);
    if (err == OS_ERR_NONE) {
       /* Можно проверять поля структуры sem_data */
    }
  }
       ...
}}
```

OSSemSet()

Описание – функция `OSSemSet()` позволяет установить значение счетчика *семафора* в желаемое состояние. Если счетчик имеет значение 0, то его значение будет изменено лишь в случае, если нет *Задач*, ожидающих этого *семафора*. Данная функция может использоваться, если *семафор* применяется в качестве «сигнального механизма»

Файл - `os_sem.c`.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_SEM_EN && OS_SEM_SET_EN>0.

Прототип void OSSemSet(OS_EVENT *pevent,
INT16U cnt,
INT8U *perr);

Аргументы:

`pevent` указатель на *семафор*. Этот указатель возвращает функция `OSSemCreate()` при создании *семафора*.

`cnt` значение, в которое следует установить счетчик *семафора*.

`perr` указатель на переменную, в которую будет возвращен код завершения, одно из следующих значений:

`OS_ERR_NONE`	если значение счетчика успешно переустановлено.
`OS_ERR_EVENT_TYPE`	если указатель `pevent` указывает не на *семафор*.
`OS_ERR_PEVENT_NULL`	если указатель `pevent` имеет значение `NULL`.
`OS_ERR_TASK_WAITING`	если имеются *Задачи*, ожидающие *семафора* (значение счетчика равно 0 и не было изменено).

Возвращаемое значение – нет.

Замечания 1) НЕ следует использовать эту функцию, если *семафор* используется для доступа к разделяемому ресурсу.

Пример использования.

```
OS_EVENT *SignalSem;

void Task (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
  OSSemSet(SignalSem, 0, &err); /* устанавливает счетчик в 0 */
       ...
  }
}
```

OSStart()

Описание – функция `OSStart()` запускает многозадачность под `uC/OS-II`. Эта функция должна быть запущена в *`startup`* коде обязательно после вызова функции `OSInit()`.

Файл - `os_core.c`.

Вызов – только из *`startup`* кода.

Транслируется всегда.

Прототип `void OSStart(void);`

Аргументы: - нет.

Возвращаемое значение – нет.

Замечания 1) `OSInit()` обязательно должна быть вызвана до вызова `OSStart()`. В свою очередь, функцию `OSStart()` имеет смысл вызывать лишь единожды. Последующие вызовы будут немедленно завершаться без выполнения каких-либо действий.

Пример использования.

```
void main (void) {
       /* User Code */
       ...
  OSInit(); /* Initialize µC/OS-II */
       /* User Code */
       ...
  OSStart(); /* Start Multitasking */
       /* Any code here should NEVER be executed! */
}
```

OSStatInit()

Описание – эта функция определяет значение, которого может достигнуть счетчик `OSIdleCtr`, инкрементируемый от значения 0 в «*пустой Задаче*». Функция `OSStatInit()` обязательно должна вызываться из первой созданной *задачи*, когда никаких других *задач* еще не создано. Найденное значение счетчика используется в последующем для определения статистики использования (степени загрузки) процессора.

Файл - `os_core.c`.

Вызов – только из `startup` кода.

Транслируется, если OS_TASK_STAT_EN && OS_TASK_CREATE_EXT_EN>0.

Прототип void OSStatInit(void);

Аргументы - нет.

Возвращаемое значение – нет.

Замечания – нет.

Пример использования. Функцию следует вызывать из *первичной Задачи*

```
void FirstAndOnlyTask (void *p_arg) {
       ...
  OSStatInit(); /* оценка загрузки процессора при неработающих Задачах */
       ...
  OSTaskCreate(_); /* создание пользовательских Задач */
  OSTaskCreate(_);
       ...
  for (;;) {
       ...
  }
}
```

OSTaskChangePrio()

Описание – функция изменяет уровень приоритета *Задачи*.

Файл - `os_task.c`.

Вызов – только из *Задачи*.

Транслируется, если OS_TASK_CHANGE_PRIO_EN>0.

Прототип INT8U OSTaskChangePrio(INT8U oldprio,
INT8U newprio);

Аргументы:

`oldprio` новое значение *приоритета*.

`newprio` старое значение *приоритета*.

Возвращаемое значение – код завершения, один из следующих:

`OS_ERR_NONE` если уровень приоритета *Задачи* успешно изменен.

`OS_PRIO_INVALID` если величина `oldprio` или `newprio` равна или превосходит значение `OS_LOWEST_PRIO`.

`OS_PRIO_EXIST` если заданный `newprio` принадлежит другой *Задаче*.

`OS_PRIO_ERR` если не существует *Задачи* с заданным `oldprio`.

`OS_TASK_NOT_EXITS` если *задача* находится на уровне приоритета `Mutex PIP`.

Замечания 1) назначаемый уровень `newprio` не должен принадлежать какой-то другой *Задаче*,
2) *Задача*, чей приоритет требуется изменить, должна существовать.

Пример использования.

```
void TaskX (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
  err = OSTaskChangePrio(10, 15);
       ...
  }
}
```

OSTaskCreate()

Описание – функция `OSTaskCreate()` создает *Задачу*, которая будет работать под управлением `uC/OS-II`. *Задачи* можно создавать либо до запуска многозадачности (до вызова `OSStart()`), либо из ранее уже созданной *задачи*. *Задачу* нельзя создать из *обработчика прерывания*. *Задача* должна представлять собой Си-функцию, которая никогда не завершается выходом через закрывающую скобку функции (либо содержит бесконечный цикл, либо уничтожает сама себя вызовом функции `OSTaskDel()`). Функция `OSTaskCreate()` оставлена наряду с более мощной функцией `OSTaskCreateExt()` для обратной совместимости, а также для использования, когда не требуются расширенные возможности функции `OSTaskCreateExt()`, а необходимо уменьшить размер программного кода.

Далее приведена типовая структура кода *задачи*, в том виде, который дан в справочнике Лаброса.

```
void Task (void *p_arg) {
  /* Do something with 'pdata' */
  for (;;) { /* Task body, always an infinite loop. */
          ...
/* Must call one of the following services: */
/* OSMboxPend() */
/* OSFlagPend() */
/* OSMutexPend() */
/* OSQPend() */
/* OSSemPend() */
/* OSTimeDly() */
/* OSTimeDlyHMSM() */
/* OSTaskSuspend() (Suspend self) */
/* OSTaskDel() (Delete self) */
          ...
  }
}
```

Следует отметить, что, вопреки комментарию в примере Лаброса, *задача* НЕ ОБЯЗАТЕЛЬНО должна содержать бесконечный цикл, вместо этого *она* может вызывать функцию `OSTaskDel()` и уничтожать сама себя !!!

Файл - `os_task.c`.

Вызов – из *задачи* или из *start-up* кода.

Транслируется, если `OS_TASK_CREATE_EN>0`. Можно запретить компиляцию кода функции `OSTaskCreate()`, если используется функция `OSTaskCreateExt()`. Одна из этих двух функций обязательно должна быть оттранслирована, иначе не создать *задач*.

Прототип INT8U OSTaskCreate(void (*task)(void *pd),
void *pdata,
OS_STK *ptos,
INT8U prio);

Аргументы:

`task` указатель на точку входа в *задачу*.

pdata указатель на (необязательную) область данных для передачи параметров в *Задачу* при ее создании. Это, в частности, позволяет создавать несколько *Задач*, используя один и тот же код функции, но передавая разным создаваемым *Задачам* различные наборы параметров.

ptos указатель на начало области памяти, которая предназначена для стека создаваемой *Задачи*. Индивидуальный стек для каждой *Задачи* требуется потому, что он будет использоваться *ядром* для сохранения контекста этой *Задачи* при ее вытеснении с процессора. Размер стека для *Задачи* определяется ее особенностями и в uC/OS-II может быть задан для каждой *Задачи* индивидуально. При задании адреса начала стека следует учитывать направление роста стека (что специфично для различных процессоров).

prio уровень приоритета создаваемой *Задачи*. Каждой *задаче* должен быть назначен индивидуальный уровень приоритета (в пределах от 1 до 62, а начиная с версии 2.80 от 1 до 254), причем, чем меньше значение prio тем выше приоритет.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE если *Задача* успешно создана.

OS_PRIO_EXIST если заданный уровень приоритета уже использован.

OS_PRIO_INVALID если заданный уровень приоритета больше, чем OS_LOWEST_PRIO.

OS_NO_MORE_TCB если нет свободных *блоков управления Задачами*.

OS_ERR_TASK_CREATE_ISR если сделана попытка создать *Задачу* из *обработчика прерывания.*

Замечания 1) для стека *Задачи* следует объявлять массив элементов типа OS_STK.

2) Каждая созданная *задача* обязательно должна вызывать один из сервисов ОС, которые «заставляют» *задачу* ожидать либо истечения некоторого интервала времени, либо наступления некоторого события (изменения состояния *почтового ящика, мьютекса, семафора, очереди, флага*). Это позволяет другим *Задачам* получать часть процессорного времени.

3) Не рекомендуется использовать уровни приоритета 0, 1, 2, 3, OS_LOWEST_PRIO-3, OS_LOWEST_PRIO-2, OS_LOWEST_PRIO-1, и OS_LOWEST_PRIO, поскольку будущие версии ОС могут использовать эти уровни приоритета.

Пример 1

1) Этот пример не использует возможности передачи параметров с использованием указателя pdata. В примере предполагается, что стек растет в сторону уменьшения адресов, поэтому в качестве адреса области стека передается старший адрес этой области. Если бы стек рос в сторону уменьшения адресов, нужно было бы передать значение &Task1Stk[0].

В этом (и в некоторых других примерах) код содержит операторы типа (void)p_arg; или p_arg= p_arg; - это делается для того, чтобы избежать предупреждений (warnings) компилятора вроде 'WARNING - variable pdata not used' – правда, далеко не все компиляторы это делают.

```
OS_STK Task1Stk[1024];

void main (void) {
  INT8U err;
       ...
  OSInit(); /* Initialize µC/OS-II */
       ...
  OSTaskCreate(     Task1,
                    (void *)0,
                    &Task1Stk[1023],
                    25);
       ...
  OSStart(); /* Start Multitasking */
}
```

```
void Task1 (void *p_arg) {
   (void)p_arg; /* Prevent compiler warning */
  for (;;) {
       /* Task code */
       ...
  }
}
```

Пример 2

Можно использовать один и тот же код функции для создания нескольких *Задач*. Например, разным *Задачам*, которые работают с асинхронным последовательным портом, можно передавать разные адреса областей, содержащих данные, подлежащие передаче, эти *Задачи* могут использовать разные скорости передачи в порте, и т.п. Помните, что каждая из этих *Задач* имеет собственный стек и собственный уровень приоритета.

```
OS_STK *Comm1Stk[1024];
COMM_DATA Comm1Data; /* структура данных, содержащая настройки UART */
       /* для одного канала */
OS_STK *Comm2Stk[1024];
COMM_DATA Comm2Data; /* структура данных, содержащая настройки UART */
       /* для второго канала */

void main (void) {
  INT8U err;
.
  OSInit(); /* Initialize µC/OS-II */
       ...
       /* создание Задачи, управляющей портом COM1 */
  OSTaskCreate(     CommTask,
                    (void *)&Comm1Data,
                    &Comm1Stk[1023],
                    25);
       /* создание Задачи, управляющей портом COM2 */
  OSTaskCreate(     CommTask,
                    (void *)&Comm2Data,
                    &Comm2Stk[1023],
                    26);
       ...
  OSStart(); /* Start Multitasking */
}

void CommTask (void *p_arg) /* код функции, управляющей портом UART */
{
COMM_DATA *pC;      /* указатель для доступа к параметрам порта */
  pC=(COMM_DATA *)p_arg;
             /* используя указатель pC, можно читать поля структуры*/
  for (;;) {
. /* Task code */
       ...
  }
}
```

OSTaskCreateExt()

Описание – функция `OSTaskCreateExt()` создает *Задачу* которая будет работать под управлением `uC/OS-II`. Эта функция действует аналогично функции `OSTaskCreate()`, кроме того, она позволяет предоставить *операционной системе* дополнительную информацию о созданной *Задаче*. *Задачи* можно создавать либо до старта многозадачности, либо из ранее созданной *Задачи*. Не разрешается создавать *Задачу* из *обработчика прерывания*. *Задача* должна представлять собой Си-функцию, которая никогда не завершается выходом через закрывающую скобку функции (либо содержит бесконечный цикл, либо уничтожает сама себя вызовом функции `OSTaskDel()`).

Первые четыре аргумента функции `OSTaskCreateExt()` в точности такие же, как в более «старой» функции `OSTaskCreate()`. Это сделано для упрощения перехода на использование новой более мощной функции `OSTaskCreateExt()`. Весьма рекомендуется в новых разработках использовать функцию `OSTaskCreateExt()` вместо старой `OSTaskCreate()`.

Функция `OSTaskCreate()` оставлена наряду с более мощной функцией `OSTaskCreateExt()` для обратной совместимости, а также для использования, когда не требуются расширенные возможности функции `OSTaskCreateExt()`, а необходимо уменьшить размер программного кода.

Файл - `os_task.c`.

Вызов – из *Задачи* или из `startup` кода.

Транслируется, если OS_TASK_CREATE_EXT_EN>0.

Прототип:
```
INT8U OSTaskCreateExt( void (*task)(void *pd),
                       void *pdata,
                       OS_STK *ptos,
                       INT8U prio,
                       INT16U id,
                       OS_STK *pbos,
                       INT32U stk_size,
                       void *pext,
                       INT16U opt);
```

Аргументы:

`task` указатель на точку входа в *задачу*.

`pdata` указатель на (необязательную) область данных для передачи параметров в *Задачу* при ее создании. Это, в частности, позволяет создавать несколько *Задач*, используя один и тот же код функции, но передавая разным создаваемым *Задачам* различные наборы параметров.

`ptos` указатель на начало области памяти, которая предназначена для *стека* создаваемой *Задачи*. Индивидуальный стек для каждой *Задачи* требуется потому, что он будет использоваться *ядром* для сохранения контекста этой *Задачи* при ее вытеснении с процессора. Размер стека для *Задачи* определяется ее особенностями и в `uC/OS-II` может быть задан для каждой *Задачи* индивидуально. При задании адреса начала стека следует учитывать направление роста стека (что специфично для различных процессоров).

`prio` уровень приоритета создаваемой *Задачи*. Каждой *Задаче* должен быть назначен индивидуальный уровень приоритета (в пределах от 1 до 62, а начиная с версии 2.80 от 1 до 254), причем, чем меньше значение `prio` тем выше приоритет.

`id` идентификационный номер *Задачи*. В настоящее время (версия 2.86) этот параметр не используется и добавлен в функцию `OSTaskCreateExt()` для будущих расширений. Рекомендуется задавать идентификационный номер таким же, как уровень приоритета *Задачи*. (В данной и более ранних версиях в качестве уникального идентификатора *Задачи* используется ее *уровень приоритета*, однако в программах, где используется динамическое изменение приоритета, это может вызвать затруднения и быть источником ошибок. Уникальный идентификатор *Задачи* был бы удобнее. – прим. ред. русского перевода.)

`pbos` указатель на «дно» стека. Если конфигурационная константа `OS_STK_GROWTH` установлена в 1, предполагается, что стек растет в сторону уменьшения адресов; таким образом, `pbos` должен указывать на

элемент стека с наименьшим адресом. Если OS_STK_GROWTH установлен в 0, предполагается, что стек растет в сторону возрастания адресов; при этом, pbos должен указывать на элемент стека с наибольшим адресом. must point to the highest valid stack location. pbos is used by the stack-checking function OSTaskStkChk().

stk_size задает размер стека *Задачи*, измеряемый в элементах. Если OS_STK имеет тип INT8U, тогда stk_size задает размер стека в байтах. Если OS_STK имеет тип INT16U, тогда stk_size задает количество 16-битовых элементов стека. Наконец, если OS_STK имеет тип INT32U, тогда stk_size задает число 32-битовых элементов стека.

pext указатель на определяемый пользователем элемент данных (обычно структуру), используемый как расширение *блока управления Задачей*. Например, такая структура может использоваться для хранения содержимого регистров плавающей точки (регистров сопроцессора, если таковой имеется в конкретной системе) при переключении контекста, время, которое каждая *Задача* занимает процессор и т.п.

opt содержит битовые поля, характеризующие опции, специфичные для данной *Задачи*. Младшие 8 битов параметра зарезервированы для нужд µC/OS-II, старшие 8 битов можно использовать для нужд приложения. Каждое битовое поле, относящееся к одной опции, может содержать один или несколько битов. Опция считается выбранной (включенной), если соответствующие биты установлены. Текущая версия µC/OS-II поддерживает следующие опции:

OS_TASK_OPT_NONE	никакие опции не заданы.
OS_TASK_OPT_STK_CHK	разрешен контроль стека *Задачи*.
OS_TASK_OPT_STK_CLR	стек *Задачи* должен быть заполнен нулями.
OS_TASK_OPT_SAVE_FP	должны сохраняться регистры плавающей точки (эта опция действует лишь при наличии в системе аппаратно реализованного сопроцессора плавающей точки.

Для получения информации о других опциях исследуйте файл ucos_ii.h.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	если *Задача* успешно создана.
OS_PRIO_EXIST	если заданный уровень приоритета уже использован.
OS_PRIO_INVALID	если заданный уровень приоритета больше, чем OS_LOWEST_PRIO.
OS_NO_MORE_TCB	если нет свободных *блоков управления Задачами*.
OS_ERR_TASK_CREATE_ISR	если сделана попытка создать *задачу* из *обработчика прерывания*.

Замечания 1) для стека *Задачи* следует объявлять массив элементов типа OS_STK.

2) Каждая созданная *Задача* обязательно должна вызывать один из сервисов ОС, которые «заставляют» *Задачу* ожидать либо истечения некоторого интервала времени, либо наступления некоторого события (изменения состояния *почтового ящика, мьютекса, семафора, очереди, флага*). Это позволяет другим *Задачам* получать часть процессорного времени.

3) Не рекомендуется использовать уровни приоритета 0, 1, 2, 3, OS_LOWEST_PRIO-3, OS_LOWEST_PRIO-2, OS_LOWEST_PRIO-1, и OS_LOWEST_PRIO, поскольку будущие версии ОС могут использовать эти уровни приоритета.

Пример 1.

E1(1) Блок управления *Задачей* расширен, используется определенная пользователем структура данных OS_TASK_USER_DATA, которая в данном случае содержит имя *Задачи* а также ряд других полей.

E1(2) Поле имени *Задачи* инициализируется с использованием стандартной библиотечной функции strcpy().

E1(4) Для данной *Задачи* разрешен контроль состояния стека, поэтому можно пользоваться функцией OSTaskStkChk().

E1(3) Подразумевается, что для используемого процессора стек растет в сторону уменьшения адресов (и, соответственно, константа OS_STK_GROWTH установлена в 1; TOS и BOS обозначают соответственно верхушку и дно стека).

```
OS_STK *TaskStk[1024];

void main (void) {
  INT8U err;
.
  OSInit(); /* Initialize µC/OS-II */
       ...
  err = OSTaskCreateExt(Task,
                        (void *)0,
                        &TaskStk[0], /* Stack grows up (TOS) */ (1)
                        10,
                        10,
                        &TaskStk[1023], /* Stack grows up (BOS) */ (1)
                        1024,
                        (void *)0,
  OS_TASK_OPT_STK_CHK); /* Stack checking enabled */ (2)
       ...
  OSStart(); /* Start Multitasking */
}

void Task (void *p_arg) {
  (void)p_arg; /* Avoid compiler warning */
  for (;;) {
       /* Task code */
       ...
  }
}
```

OSTaskDel()

Описание – функция `OSTaskDel()` удаляет *Задачу* с заданным уровнем приоритета. Вызывающая функцию *Задача* также может удалить сама себя, задав значение параметра `prio` равным `OS_PRIO_SELF`.

Файл - `os_task.c`.

Вызов – только из *Задачи*.

Транслируется, если `OS_TASK_DEL_EN`>0.

Прототип INT8U OSTaskDel(INT8U prio);

Аргументы:

`prio` уровень приоритета удаляемой *Задачи*. Вызывающая функцию *Задача* также может удалить сама себя, задав значение параметра `prio` равным `OS_PRIO_SELF`.

Возвращаемое значение – код завершения, одно из следующих значений:

`OS_ERR_NONE` если удаление *Задачи* выполнено успешно.

`OS_TASK_DEL_IDLE` если сделана попытка удаления *пустой Задачи*, это недопустимо, и удаление не выполняется.

`OS_TASK_DEL_ERR` если *Задачи* с заданным уровнем приоритета не существует.

`OS_PRIO_INVALID` если задано значение `prio` большее, чем `OS_LOWEST_PRIO`.

`OS_TASK_DEL_ISR` если делается попытка удалить *Задачу* из *обработчика прерывания*.

`OS_TASK_NOT_EXIST` если задано значение `prio` соответствующее `Mutex PIP`.

Замечания 1) Функция проверяет, что не делается попытки удалить *пустую Задачу*.

2) Следует быть аккуратным при удалении, так как *Задача* может владеть ресурсами которые после удаления останутся в состоянии «занято»). Более надежно использовать функцию `OSTaskDelReq()`.

Пример использования.

```
void TaskX (void *p_arg) {
  INT8U err;
  for (;;) {
       ...
    err = OSTaskDel(10);   /* удалить Задачу с приоритетом 10 */
    if (err == OS_ERR_NONE) {
       ...                          /* Задача была удалена */
    }
       ...
  }
}
```

OSTaskDelReq()

Описание – функция OSTaskDelReq() позволяет послать *Задаче* запрос на самоудаление. Эту функцию используют, если есть основания предполагать, что *Задача*, подлежащая удалению, может владеть объектами синхронизации или коммуникации (*мьютексами, семафорами, почтовыми ящиками*, и т.д.). Если требуется удаление такой *Задачи*, то какая-либо другая *Задача* должна вызвать функцию OSTaskDelReq() с параметром, равным приоритету *Задачи*, подлежащей удалению. Последняя, получив управление, также должна вызвать функцию OSTaskDelReq(OS_PRIO_SELF) и проанализировать код завершения. Если он равен OS_TASK_DEL_REQ, это значит, что *Задача*, подлежащая удалению, должна освободить все занятые ею ресурсы, а затем удалить сама себя. Такая техника обеспечивает гарантированное освобождение *Задачей* ресурсов перед ее удалением.

Задача, запросившая удаление, затем может проверить, было ли оно выполнено. Для этого следует снова вызвать функцию OSTaskDelReq() с параметром, равным приоритету *задачи*, подлежавшей удалению, и проанализировать код завершения. Если он равен OS_TASK_NOT_EXIST, это свидетельство того, что удаление успешно произошло.

Файл - os_task.c.

Вызов – только из *Задачи*.

Транслируется, если OS_TASK_DEL_EN>0.

Прототип INT8U OSTaskDelReq(INT8U prio);

Аргументы:

prio приоритет *Задачи*, подлежащей удалению. *Задача* может удалить сама себя, задав значение prio равным OS_PRIO_SELF.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	если запрос на удаление был успешно зарегистрирован.
OS_TASK_NOT_EXIST	*задачи* с заданным приоритетом не существует.
OS_TASK_DEL_IDLE	запрос на удаление *«пустой» задачи* (недопустимо, запрос не регистрируется!).
OS_PRIO_INVALID	задан уровень приоритета больший, чем OS_LOWEST_PRIO.
OS_TASK_DEL_REQ	был зарегистрирован запрос на удаление *задачи* с указанным приоритетом.

Замечания 1) Функция OSTaskDelReq() проверяет попытку удаления *«пустой» задачи*.

Пример использования.

```
void TaskThatDeletes (void *p_arg) {  /* Задача с приоритетом 5 */
INT8U err;                             /* удаляет Задачу с приоритетом 10*/
  for (;;) {
       ...
  err = OSTaskDelReq(10); /* запрос Задаче #10 самоудалиться */
    if (err == OS_ERR_NONE) {
      while (err != OS_TASK_NOT_EXIST) {
        err = OSTaskDelReq(10);
        OSTimeDly(1); /* ожидание самоудаления Задачи #10 */
      }       /* дождались! */
    }
       ...
  }
}

void TaskToBeDeleted (void *p_arg) {  /* удаляемая Задача, ее приоритет 10 */
  for (;;) {
    OSTimeDly(1);
    if (OSTaskDelReq(OS_PRIO_SELF) == OS_TASK_DEL_REQ) {
      /* освободить все занятые ресурсы; */
      OSTaskDel(OS_PRIO_SELF);
    }
  }
```

```
}
```

OSTaskNameGet()

Описание – функция OSTaskNameGet() позволяет получить значение имени, которое было ранее назначено *Задаче*. Имя представляет собой ASCII-строку, размер которой не превышает OS_TASK_NAME_SIZE символов, включая нуль-терминатор.

Файл - os_task.c.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_TASK_NAME_SIZE>0.

Прототип INT8U OSTaskNameGet(INT8U prio,
INT8U *pname,
INT8U *perr);

Аргументы:

prio приоритет *Задачи*, имя которой требуется получить. Можно узнать имя текущей исполняемой *Задачи*, задав значение параметра равным OS_PRIO_SELF.

pname указатель на ASCII-строку, в которую будет возвращено значение имени. Длина строки должна быть не меньше OS_TASK_NAME_SIZE байтов, включая нуль-терминатор.

perr указатель на переменную, куда будет помещен код завершения, одно из следующих значений:

OS_ERR_NONE	успешное завершение, имя *Задачи* скопировано в *pname.
OS_TASK_NOT_EXIST	*Задачи* с заданным приоритетом не существует.
OS_PRIO_INVALID	заданный приоритет больше, чем OS_LOWEST_PRIO.
OS_ERR_PNAME_NULL	указатель pname имеет значение NULL.

Возвращаемое значение – размер возвращенной ASCII-строки, либо 0 в случае ошибки.

Замечания 1) *Задача* должна быть создана перед использованием данной функции.

2) Следует обеспечить достаточное место в строке-приемнике для возвращаемого имени.

Пример использования.

```
INT8U EngineTaskName[30];

void Task (void *p_arg)  {
  INT8U err;
  INT8U size;
  for (;;) {
    size = OSTaskNameGet(OS_PRIO_SELF, &EngineTaskName[0], &err);
       ...
  }
}
```

OSTaskNameSet()

Описание – функция OSTaskNameSet() позволяет задать *Задаче* имя, которое представляет собой ASCII-строку, размер которой не превышает OS_TASK_NAME_SIZE символов, включая нуль-терминатор.

Файл - os_task.c.

Вызов – из Задачи или из обработчика прерывания.

Транслируется, если OS_TASK_NAME_SIZE>0.

Прототип void OSTaskNameSet(INT8U prio,
INT8U *pname,
INT8U *perr);

Аргументы:

prio приоритет *Задачи*, имя которой требуется получить. Можно узнать имя исполняемой *Задачи*, задав значение параметра равным OS_PRIO_SELF.

pname указатель на ASCII-строку, в которой задано значение имени. Длина строки должна быть не меньше OS_TASK_NAME_SIZE байтов, включая нуль-терминатор.

perr указатель на переменную, куда будет помещен код завершения, одно из следующих значений:

OS_ERR_NONE успешное завершение, имя *задачи* задано.
OS_TASK_NOT_EXIST *задачи* с заданным приоритетом не существует.
OS_PRIO_INVALID заданный приоритет больше, чем OS_LOWEST_PRIO.
OS_ERR_TASK_NAME_TOO_LONG если длина заданного имени превосходит величину OS_TASK_NAME_SIZE.
OS_ERR_PNAME_NULL указатель pname имеет значение NULL.

Возвращаемое значение – нет.

Замечания 1) *Задача* должна быть создана до использования любой функции, работающей с ней.

Пример использования.

```
void Task (void *p_arg) {
  INT8U err;
  for (;;) {
    OSTaskNameSet(OS_PRIO_SELF, "Engine Task", &err);
       ...
  }
}
```

OSTaskResume()

Описание – функция OSTaskResume() возобновляет выполнение *Задачи*, приостановленной функцией OSTaskSuspend().

Файл - os_task.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_TASK_SUSPEND_EN>0.

Прототип INT8U OSTaskResume(INT8U prio);

Аргументы:

prio задает приоритет *Задачи*, которую следует возобновить.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	если *Задача* возобновлена успешно.
OS_TASK_RESUME_PRIO	если *Задачи* с заданным приоритетом не существует.
OS_TASK_NOT_SUSPENDED	если *Задача* с указанным приоритетом не была приостановлена.
OS_PRIO_INVALID	если значение prio больше или равно OS_LOWEST_PRIO.
OS_TASK_NOT_EXIST	указанная *Задача* находится на уровне приоритета Mutex PIP.

Замечания – нет.

Пример использования.

```
void TaskX (void *p_arg)  {
  INT8U err;
  for (;;) {
       ...
    err = OSTaskResume(10); /* Resume task with priority 10 */
    if (err == OS_ERR_NONE) {
       /* Task was resumed */
       ...
    }
       ...
  }
}
```

OSTaskStkChk()

Описание – функция OSTaskStkChk() определяет статистику использования стека *Задачи* – величину свободного места в стеке, а также величину использованного пространства в стеке. Для использования этой функции *Задача* должна быть создана с помощью функции OSTaskCreateExt() и задано значение OS_TASK_OPT_STK_CHK для аргумента opt.

Размер стека определяется путем подсчета «нулей» в стеке, начиная от его конца до первого ненулевого элемента. Естественно, это предполагает, что изначально (при создании *Задачи*) все пространство стека заполнено нулевыми значениями. Для обеспечения этого следует задать опцию OS_TASK_OPT_STK_CLR=1 при создании *Задачи*. Если startup код обнуляет содержимое памяти, и в приложении никогда не выполняется уничтожение *Задач*, можно установить опцию OS_TASK_OPT_STK_CLR=0,. Это позволяет уменьшить время выполнения функции OSTaskCreateExt().

Файл - os_task.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_TASK_CREATE_EXT>0.

Прототип `INT8U OSTaskStkChk(INT8U prio, OS_STK_DATA *p_stk_data);`

Аргументы:

prio приоритет *Задачи*, для которой требуется выполнить проверку состояния стека. *Задача* может получить информацию о собственном стеке, задав значение этого параметра равным OS_PRIO_SELF.

P_stk_data указатель на структуру типа OS_STK_DATA, которая содержит следующие поля:

```
INT32U OSFree; /* Количество свободных байтов в стеке */
INT32U OSUsed; /* Количество занятых байтов в стеке   */
```

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	если вызов функции завершился успешно.
OS_PRIO_INVALID	если задано значение приоритета больше, чем OS_LOWEST_PRIO.
OS_TASK_NOT_EXIST	если *Задачи* с заданным приоритетом не существует.
OS_TASK_OPT_ERR	если не задана опция OS_TASK_OPT_STK_CHK при создании *задачи* посредством OSTaskCreateExt() или если *задача* создана при помощи OSTaskCreate().
OS_ERR_PDATA_NULL	если p_stk_data имеет значение NULL.

Замечания 1) Время выполнения этой функции зависит от размера стека *Задачи* и поэтому точно не определено.

2) Приложение может определить размер стека суммированием двух величин .OSFree и .OSUsed из структуры OS_STK_DATA.

3) Вообще-то, эта функция может вызываться из *обработчика прерывания*, однако следует учитывать неопределенность времени ее выполнения (и поэтому так делать не рекомендуется).

Пример использования.

```
void Task (void *p_arg)  {
  OS_STK_DATA stk_data;
  NT32U stk_size;
  for (;;) {
       ...
    err = OSTaskStkChk(10, &stk_data);
    if (err == OS_ERR_NONE) {
      stk_size = stk_data.OSFree + stk_data.OSUsed;
    }
       ...
  }
}
```

OSTaskSuspend()

Описание – функция OSTaskSuspend() принудительно приостанавливает (блокирует) *Задачу*. Вызывающая *Задача* может приостановить сама себя, задав значение параметра (приоритета) равным OS_PRIO_SELF (при этом *ядро* подставляет значение приоритета выполняемой *Задачи*). Возобновить приостановленную этой функцией *Задачу* может только другая *Задача* вызовом функции OSTaskResume(). Функции приостановки *Задачи* аддитивны в том смысле, что если *Задача* приостановлена функцией OSTimeDly(n) на n тиков, а затем выполнена функция OSTaskSuspend() для этой же *Задачи*, последняя сможет возобновиться лишь если время задержки вышло и была вызвана функция OSTaskResume(). То же имеет место при ожидании семафора *Задачей*, приостановленной с помощью OSTaskSuspend() – оба условия приостановки должны быть аннулированы для того, чтобы *Задача* могла продолжить выполнение.

Файл - os_task.c.

Вызов – только из *Задачи*.

Транслируется, если OS_TASK_SUSPEND_EN>0.

Прототип INT8U OSTaskSuspend(INT8U prio);

Аргументы:

prio задает приоритет *Задачи*, которую следует приостановить.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE если вызов функции завершился успешно.
OS_TASK_SUSPEND_IDLE если сделана попытка приостановить *пустую Задачу*, это недопустимо и не выполняется.
OS_PRIO_INVALID значение prio превышает величину OS_LOWEST_PRIO.

OS_TASK_SUSPEND_PRIO *Задачи* с уровнем приоритета prio не существует.

OS_TASK_NOT_EXITS указанная *Задача* находится на уровне приоритета Mutex PIP.

Замечания 1) Функции OSTaskSuspend() и OSTaskResume() должны использоваться «в паре».
2) *Задача*, приостановленная с помощью OSTaskSuspend(), может быть возобновлена только с помощью OSTaskResume().

Пример использования.

```
void TaskX (void *p_arg)  {
  INT8U err;
  for (;;) {
       ...
    err = OSTaskSuspend(OS_PRIO_SELF); /* Suspend current task */
       /* Execution continues when ANOTHER task explicitly resumes this task. */
       ...
  }
}
```

OSTaskQuery()

Описание – функция OSTaskQuery() позволяет получить информацию о *Задаче*. Следует объявить структуру данных типа OS_TCB, в которую функция OSTaskQuery() при ее вызове скопирует содержимое *блока управления Задачей*. Следует быть аккуратным при использовании полученных данных, в особенности, указателей OSTCBNext и OSTCBPrev, поскольку они поддерживают текущий список *блоков управления Задачей*.

Файл - os_task.c.

Вызов – из Задачи или из обработчика прерывания

Транслируется, если OS_TASK_QUERY_EN>0.

Прототип INT8U OSTaskQuery(INT8U prio,
OS_TCB *p_task_data);

Аргументы:

prio задает приоритет *Задачи*, о которой запрашивается информация.

p_task_data указатель на структуру типа OS_TCB, в которую будет скопировано содержимое *блока управления Задачей*.

Возвращаемое значение – код завершения, одно из следующих значений:

OS_ERR_NONE	если вызов функции завершился успешно.
OS_PRIO_INVALID	значение prio превышает величину OS_LOWEST_PRIO.
OS_PRIO_ERR	*задачи* с уровнем приоритета prio не существует.
OS_TASK_NOT_EXITS	указанная *задача* находится на уровне приоритета Mutex PIP.
OS_ERR_PDATA_NULL	указатель p_task_data имеет значение NULL.

Замечания 1) Набор полей в *блоке управления Задачей* зависит от следующих опций в os_cfg.h):

- OS_TASK_CREATE_EN
- OS_Q_EN
- OS_FLAG_EN
- OS_MBOX_EN
- OS_SEM_EN
- OS_TASK_DEL_EN

Пример использования.

```
void Task (void *p_arg)  {
  OS_TCB task_data;
  INT8U err;
  void *pext;
  INT8U status;
  for (;;) {
       ...
    err = OSTaskQuery(OS_PRIO_SELF, &task_data);
    if (err == OS_ERR_NONE) {
      pext = task_data.OSTCBExtPtr; /* Get TCB extension pointer */
      status = task_data.OSTCBStat; /* Get task status */
       ...
    }
       ...
  }
}
```

OSTimeDly()

Описание – функция `OSTimeDly()` позволяет *Задаче* приостановить самой себя на заданное параметром функции количество тиков системного таймера. Максимально допустимое значение параметра составляет 65535. Нулевое значение параметра означает отсутствие приостановки, функция `OSTimeDly()` немедленно завершается, а вызвавшая ее *Задача* продолжает работу. Если заданное количество тиков больше нуля, *ядро* выполняет операции *планирования* и *диспетчеризации* (и переключает контекст, если есть *Задачи*, готовые к выполнению). Реальная продолжительность приостановки зависит от частоты системных тиков (см. значение `OS_TICKS_PER_SEC` в файле конфигурации `os_cfg.h`).

Файл - `os_time.c`.

Вызов – только из *Задачи*.

Транслируется – всегда.

Прототип void OSTimeDly(INT16U ticks);

Аргументы:

`ticks` количество тиков системного таймера, на которое следует приостановить *Задачу*.

Возвращаемое значение – нет.

Замечания 1) При значении параметра 0 функция немедленно завершается и приостановки не происходит.

2) Для гарантированной приостановки на время «не менее» желаемого количества тиков, значение параметра следует задавать на единицу больше желаемого значения.

Пример использования.

```
void TaskX (void *p_arg)  {
  for (;;) {
       ...
  OSTimeDly(10); /*Задерживает Задачу на время в интервале от 9 до 10 тиков*/
       ...
  }
}
```

OSTimeDlyHMSM()

Описание – функция `OSTimeDlyHMSM()` приостанавливает *Задачу* на интервал времени, задаваемый в часах, минутах, секундах и миллисекундах. Этот формат более привычен для программиста, нежели тики системного таймера. Если хотя бы один из параметров не равен нулю, текущая *Задача* приостанавливается, выполняется планирование, и переключение контекста.

Файл - `os_time.c`.

Вызов – из Задачи или из обработчика прерывания

Транслируется – всегда.

Прототип void OSTimeDlyHMSM (INT8U hours,
INT8U minutes,
INT8U seconds,
INT8U milli);

Аргументы:

`hours` составляющая времени приостановки в часах (от 0 до 255).
`minutes` составляющая времени приостановки в минутах (от 0 до 59).
`seconds` составляющая времени приостановки в часах (от 0 до 59).
`milli` составляющая времени приостановки в часах (от 0 до 999). Разрешающая способность этого параметра не лучше величины системного тика. Функция производит округление заданного значения миллисекунд до ближайшей величины, кратной системному тику. Так, если величина тика равна 10 мс, а задано значение параметра 15 мс, реально будет использовано значение 20 мс.

Возвращаемое значение – код завершения, одно из следующих значений:

`OS_ERR_NONE`	задан корректный аргумент, и вызов завершился успешно.
`OS_TIME_INVALID_MINUTES`	значение минут превышает 59.
`OS_TIME_INVALID_SECONDS`	значение секунд превышает 59.
`OS_TIME_INVALID_MS`	значение миллисекунд превышает 999.
`OS_TIME_ZERO_DLY`	все четыре аргумента равны нулю.
`OS_ERR_TIME_DLY_ISR`	функция вызвана из *обработчика прерывания* (недопустимо).

Замечания 1) Если значения всех четырех аргументов равны нулю, функция завершается немедленно, не приводя к приостановке *Задачи*. Если общее время задержки превышает 65535 тиков системного таймера, приостановленную *Задачу* нельзя будет возобновить вызовом функции `OSTimeDlyResume()` из другой *Задачи*.

Пример использования.

```
void TaskX (void *p_arg)  {
  for (;;) {
       ...
    OSTimeDlyHMSM(0, 0, 1, 0); /* Задержать Задачу TaskX на 1 секунду */
       ...
  }
}
```

OSTimeDlyResume()

Описание – функция `OSTimeDlyResume()` переводит в состояние готовности *Задачу*, приостановленную вызовом `OSTimeDly()` или `OSTimeDlyResume()`.

Файл - `os_time.c`.

Вызов – из Задачи или из обработчика прерывания

Транслируется - всегда.

Прототип INT8U OSTimeDlyResume(INT8U prio);

Аргументы:

`prio` – приоритет *задачи*, которую следует возобновить.

Возвращаемое значение – код завершения, одно из следующих значений:

`OS_ERR_NONE`	если вызов функции завершился успешно.
`OS_PRIO_INVALID`	значение `prio` превышает величину `OS_LOWEST_PRIO`.
`OS_TIME_NOT_DLY`	*Задача* не находится в состоянии *задержки*.
`OS_TASK_NOT_EXIST`	указанная *Задача* не существует, либо находится на уровне приоритета `Mutex PIP`. ??? и что будет?

Замечания 1) Вызов данной функции для *Задачи*, ожидающей события (*мьютекса*, *сообщения* и т.п.) с заданным *тайм-аутом* приведет к тому, что для *Задачи* «как-бы» завершился *тайм-аут*. Если именно этого и хочется, то так можно делать.

2) Невозможно возобновить *Задачу*, приостановленную функцией `OSTimeDlyHMSM()`, если задержка, соответствующая заданной в этой функции комбинации параметров, превышает 65535 системных тиков (0 часов + 10 минут + 55 секунд + 350 мс или больше).

Пример использования.

```
void TaskX (void *pdata)  {
  INT8U err;
  for (;;) {
       ...
    err = OSTimeDlyResume(10); /* Resume task with priority 10 */
    if (err == OS_ERR_NONE) {
       /* Task was resumed */
       ...
    }
       ...
  }
}
```

OSTimeGet()

Описание – функция `OSTimeGet()` возвращает текущее значение системного времени в виде 32-разрядного количества системных тиков, прошедших от момента времени, когда выполнился запуск системного таймера (если системное время не переустанавливалось функцией `OSTimeSet()`).

Файл - `os_time.c`.

Вызов – из Задачи или из обработчика прерывания

Транслируется – всегда.

Прототип INT32U OSTimeGet(void);

Аргументы – нет.

Возвращаемое значение – текущее значение системного времени.

Замечания - нет.

Пример использования.

```
void TaskX (void *p_arg)  {
  INT32U clk;
  for (;;) {
       ...
    clk = OSTimeGet(); /* Возвратит значение системного времени */
       ...
  }
}
```

OSTimeSet()

Описание – функция `OSTimeSet()` позволяет установить текущее значение системного времени (32-беззнаковое целое – содержимое счетчика системных тиков).

Файл - `os_time.c`.

Вызов – из Задачи или из обработчика прерывания

Транслируется – всегда.

Прототип void OSTimeSet(INT32U ticks);

Аргументы:

`ticks` новое 32-битовое беззнаковое содержимое счетчика системных тиков.

Возвращаемое значение – нет.

Замечания – нет.

Пример использования.

```
void TaskX (void *p_arg)  {
  for (;;) {
       ...
    OSTimeSet(0L); /* Сбросится в 0 системное время */
       ...
  }
}
```

OSTimeTick()

Описание – функция обрабатывает «тик» системного таймера. Основная работа этой функции состоит в проверке для всех *Задач*, не ждут ли они истечения времени: либо задержки, заданной функциями `OSTimeDly...()`, либо истечения *тайм-аута* при ожидании события. Эта функция обычно вызывается обработчиком прерывания системного таймера, но также может быть вызвана высокоприоритетной *Задачей*.

Файл - `os_core.c`.

Вызов – из Задачи или из обработчика прерывания

Транслируется – всегда.

Прототип void OSTimeTick(void);

Аргументы: - нет.

Возвращаемое значение – нет.

Замечания 1) Время выполнения этой функции прямо пропорционально количеству созданных в приложении *Задач*. Если данная функция вызывается из *Задачи*, то ее приоритет должен быть очень высоким, потому что данная функция ответственна за подсчет времен задержек и *тайм-аутов*.

Пример использования.

```
void  Tmr_TickISR_Handler (void) {
    TIMER0->clear   = 0x00000000L;  /* Сброс системного таймера */
    OSTimeTick();  /* Вызов функции uC/OS-II обработки тика     */
}
```

OSTmrCreate()

Описание – функция OSTmrCreate() создает объект ядра *Таймер*, который можно сконфигурировать либо в непрерывный режим (значением параметра opt, равным OS_TMR_OPT_PERIODIC), либо в однократный режим значением параметра opt, равным OS_TMR_OPT_ONE_SHOT). *Таймер* декрементируется через заданный период (константа ???) и по достижении 0 может быть вызвана функция callback. Эту функцию можно использовать разными способами (например, сигнализировать *Задаче* о переполнении *Таймера*). Рекомендуется делать время исполнения функции callback минимальным.

Создание *Таймера* не приводит к его запуску, для этого следует вызвать функцию OSTmrStart(). Если *Таймер* сконфигурирован в режим однократного запуска, и досчитал до переполнения, для повторного запуска также следует использовать OSTmrStart(). Если не планируется повторный запуск однократного *Таймера*, его можно удалить (уничтожить), используя функцию OSTmrDel(). Удаление *Таймера* допускается выполнять из функции callback.

Файл - os_tmr.c.

Вызов – только из *Задачи*.

Транслируется – если OS_TMR_EN>0.

Прототип

```
OS_TMR *OSTmrCreate(INT32U              dly,
                    INT32U              period,
                    INT8U               opt,
                    OS_TMR_CALLBACK callback,
                    void *              callback_arg,
                    INT8U *             pname,
                    INT8U *             perr);
```

Аргументы:

dly задает начальную задержку ???
В режиме однократного запуска это время срабатывания

В периодическом режиме это задержка до начала первого периода.??? Единицы, в которых задается dly определяются частотой вызова функции OSTmrSignal().??? Значением константы OS_TMR_CFG_TICKS_PER_SEC. Помните, что *Таймер* НЕ запускается автоматически при его создании.

period определяет период переполнения *Таймера*. Единица времени, в которых задается параметр, равна периодичности вызова функции OSTmrSignal().

opt параметр задает один из двух режимов работы *Таймера* и может принимать лишь одно из двух следующих значений:
OS_TMR_OPT_PERIODIC - режим периодического запуска,
OS_TMR_OPT_ONE_SHOT - режим однократного запуска.

callback (необязательный параметр) адрес функции, которая будет вызываться каждый раз по переполнению создаваемого *Таймера*, либо по вызову функции OSTmrStop(), останавливающей указанный *Таймер*. (??? это требуется проверить)
Функция должна быть объявлена следующим образом:

```
void MyCallback (void *ptmr, void *callback_arg);
```

При переполнении *Таймера* эта функция будет вызвана, ей будет передан указатель на *Таймер* и параметр callback_arg (см. след. пункт).

Если не предполагается использовать вызов функции по переполнению *Таймера*, данный параметр должен иметь значение NULL.

callback_arg (необязательный параметр) указатель на блок параметров, передаваемый вызываемой по переполнению *Таймера* функции. Если параметры передавать не требует ся, данный параметр должен иметь значение NULL.

`pname` указатель на текстовую строку, которая содержит *имя* создаваемого *Таймера*. Это имя можно получить вызовом функции `OSTmrNameGet()`.

`perr` адрес переменной, в которую будет при создании *Таймера* возвращен код завершения, одно из следующих значений:

- `OS_ERR_NONE` если *Таймер* создан успешно;
- `OS_ERR_TMR_INVALID_DLY` параметр `dly` имеет значение 0 для режима однократного запуска;
- `OS_ERR_TMR_INVALID_PERIOD` параметр `period` имеет значение 0 для режима периодического запуска;
- `OS_ERR_TMR_INVALID_OPT` неверное значение параметра `opt`;
- `OS_ERR_TMR_ISR` функция создания *Таймера* вызвана из обработчика прерывания;
- `OS_ERR_TMR_NON_AVAIL` нет свободных *Таймеров* (структур данных `OS_TMR`);
- `OS_ERR_TMR_NAME_TOO_LONG` имя *Таймера* превышает значение `OS_TMR_CFG_NAME_SIZE`.

Возвращаемое значение – указатель на структуру `OS_TMR`, связанную с созданным *Таймером*, либо `NULL`, если не удалось создать *Таймер* (в этом случае следует проверять код завершения).

Замечания

1) Следует всегда проверять возвращаемое значение, чтобы быть уверенным в результате.
2) Функцию создания *Таймера* нельзя вызывать из обработчика прерывания.
3) Помните, что *Таймер* НЕ запускается автоматически при создании, для запуска следует вызвать функцию `OSTmrStart()`.

Пример использования.

```
void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
    CloseDoorTmr = OSTmrCreate( 10,
                                100,
                                OS_TMR_OPT_PERIODIC,
                                DoorCloseFnct,
                                (void *)0,
                               "Door Close",
                               &err);
    if (err == OS_ERR_NONE) {
      /* Таймер создан, но не запущен !!! */
    }
  }
}
```

OSTmrDel()

Введена в 2.81, обновлена в 2.83.

Описание – позволяет удалить *Таймер*. Если *Таймер* запущен, он будет остановлен, а затем удален. Если *Таймер* уже в состоянии останова, он просто удаляется. Удаление неиспользуемого *Таймера* является заботой программиста, если *Таймер* удален, на него не следует более ссылаться.

Файл - `os_tmr.c`.

Вызов – только из *Задачи*.

Транслируется – если `OS_TMR_EN>0`.

Прототип `BOOLEAN OSTmrDel( OS_TMR *ptmr, INT8U *perr);`

Аргументы:

`ptmr` указатель на структуру `OS_TMR`, связанную с *Таймером*, который требуется удалить. Этот указатель был получен при создании *Таймера*.

`perr` адрес переменной, в которую будет возвращен код завершения, одно из следующих значений:

`OS_ERR_NONE`	если удаление выполнено успешно;
`OS_ERR_TMR_INVALID`	если `ptmr` равен `NULL`;
`OS_ERR_TMR_INVALID_TYPE`	если `ptmr` указывает НЕ на *Таймер*;
`OS_ERR_TMR_ISR`	если вызов произведен из *обработчика прерываний*;
`OS_ERR_TMR_INACTIVE`	если `ptmr` указывает на неактивный *Таймер* (т.е. если последний еще не создан, либо уже удален).

Возвращаемое значение – одно из двух: `OS_TRUE`, если *Таймер* удален успешно, либо `OS_FALSE`, если удаление не удалось. В последнем случае следует проверять код завершения (см. предыд. пункт).

Замечания

1) Следует всегда проверять возвращаемое значение, чтобы быть уверенным в результате.
2) Функцию удаления *Таймера* нельзя вызывать из обработчика прерывания.
3) Если *Таймер* удален, НЕ СЛЕДУЕТ обращаться к его указателю??? (формулировка)

Пример использования.

```
OS_TMR *CloseDoorTmr;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
    CloseDoorTmr = OSTmrDel(CloseDoorTmr,
                            &err);
    if (err == OS_ERR_NONE) {
      /* Таймер был удален, нельзя к нему обращаться! */
    }
  }
}
```

OSTmrNameGet()

Появилась в v2.81.

Описание – позволяет получить имя, ассоциированное с *Таймером*, заданное при его создании. Имя возвращается в указанный массив символов `pdest`, который должен быть не короче величины `OS_TMR_CFG_NAME_SIZE`, заданной в конфигурационном файле `os_cfg_r.h`. ???

Файл - `os_tmr.c`.

Вызов – только из *Задачи*.

Транслируется – если `OS_TMR_EN>0`.

Прототип void OSTmrNameGet(OS_TMR * ptmr,
INT8U * pdest,
INT8U * perr);

Аргументы:

`ptmr` – указатель на *Таймер*. Возвращается при создании *Таймера* функцией `OSTmrCreate()`.

`pdest` – указатель на строку символов длиной не короче величины `OS_TMR_CFG_NAME_SIZE`, заданной в конфигурационном файле `os_cfg.h`.

`perr` – адрес переменной, в которую будет помещен код завершения, один из следующих:

`OS_ERR_NONE`	- если имя *Таймера* успешно скопировано;
`OS_ERR_TMR_INVALID_DEST`	– если аргумент `pdest` имеет значение `NULL`.
`OS_ERR_TMR_INVALID`	– если аргумент `ptmr` имеет значение `NULL`.
`OS_ERR_TMR_INVALID_TYPE`	– если аргумент `ptmr` указывает НЕ на *Таймер*.
`OS_ERR_TMR_ISR`	- если вызов произведен из обработчика прерываний;
`OS_ERR_TMR_INACTIVE`	- если `ptmr` указывает на неактивный *Таймер* (т.е. если последний еще не создан, либо уже удален).

Возвращаемое значение – длина возвращенного имени *Таймера* в символах.

Замечания 1) Длина массива-приемника должна иметь достаточный размер не менее величины `OS_TMR_CFG_NAME_SIZE`, заданной в конфигурационном файле `os_cfg_r.h`.

2) Следует всегда проверять возвращаемое значение, чтобы быть уверенным в результате.

3) Функцию нельзя вызывать из обработчика прерывания.

Пример использования.

```
INT8U CloseDoorTmrName[80];
OS_TMR *CloseDoorTmr;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
    OSTmrNameGet(CloseDoorTmr, &CloseDoorTmrName[0], &err);
    if (err == OS_ERR_NONE) {
      /* CloseDoorTmrName[] теперь содержит имя Таймера CloseDoorTmr */
    }
  }
}
```

OSTmrRemainGet()

Функция появилась в v2.81

Описание – функция возвращает значение времени, оставшееся до переполнения указанного *Таймера*. Значение представлено в единицах, задаваемых константой `OS_TMR_CFG_TICKS_PER_SEC` в файле `os_cfg_r.h`.

Файл - `os_tmr.c`.

Вызов – только из *Задачи*.

Транслируется – если `OS_TMR_EN>0`.

Прототип INT32U OSTmrRemainGet(OS_TMR * ptmr,
INT8U * perr);

Аргументы:

`ptmr` указатель на структуру , ассоциированную с *Таймером*.

`perr` адрес переменной в которую будет помещен код завершения, один из следующих:

`OS_ERR_NONE` если функция успешно возвратила значение оставшегося времени;
`OS_ERR_TMR_INVALID` – если аргумент `ptmr` имеет значение `NULL`.
`OS_ERR_TMR_INVALID_TYPE` – если аргумент `ptmr` указывает НЕ на *Таймер*;
`OS_ERR_TMR_ISR` - если вызов произведен из обработчика прерывания;
`OS_ERR_TMR_INACTIVE` - если `ptmr` указывает на неактивный *Таймер* (т.е. если последний еще не создан, либо уже удален).

Возвращаемое значение – время, оставшееся до переполнения *Таймера*, либо 0, если *Таймер* уже переполнился, либо при вызове функции произошла ошибка.

Замечания

1) Следует всегда проверять возвращаемое значение, чтобы быть уверенным в результате.
2) Функцию нельзя вызывать из обработчика прерывания.

Пример использования.

```
INT32U TimeRemainToCloseDoor;
OS_TMR *CloseDoorTmr;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
    TimeRemainToCloseDoor = OSTmrRemainGet(CloseDoorTmr, &err);
    if (err == OS_ERR_NONE) {
      /* Вызов завершился успешно */
    }
  }
}
```

OSTmrSignal()

С v2.81 ???

Описание – `OSTmrSignal()` должна вызываться ??? периодически либо из *Задачи*, либо из обработчика прерывания. Функция обеспечивает периодическое обновление значений созданных *Таймеров*. Обычное место, откуда рекомендуется вызывать функцию `OSTmrSignal()` это функция `OSTimeTickHook()`. (???*Надо вообще пояснить роль константы* `OS_TMR_CFG_TICKS_PER_SEC`)

Файл - `os_tmr.c`.

Вызов – только из *Задачи*.

Транслируется – если `OS_TMR_EN>0`.

Прототип INT8U OSTmrSignal(void);

Аргументы – нет.

Возвращаемое значение – код завершения, одна из следующих величин:

`OS_ERR_NONE` успешное завершение, сигнал на модификацию *Таймеров*.
`OS_ERR_SEM_OVF` вызовы происходят чаще, чем *Задача* `OSTmr_Task()` способна обработать их. Это свидетельство высокой загрузки системы.
`OS_ERR_EVENT_TYPE` ошибка обращения к внутреннему ??? семафору, этот и следующий коды завершения маловероятны, так как семафор создан и используется внутренним кодом μC/OS-II;
`OS_ERR_PEVENT_NULL` ошибка обращения к внутреннему ??? семафору.

Замечания - нет .

Пример использования.

```
#if OS_TMR_EN > 0
static INT16U OSTmrTickCtr = 0;
#endif

void OSTimeTickHook (void) {
  #if OS_TMR_EN > 0
  OSTmrTickCtr++;
  if (OSTmrTickCtr >= (OS_TICKS_PER_SEC / OS_TMR_CFG_TICKS_PER_SEC)) {
    OSTmrTickCtr = 0;
    OSTmrSignal();
  }
  #endif
}
```

OSTmrStart()

Описание – позволяет запустить (перезапустить) *Таймер* на счет (на убывание). *Таймер* должен быть создан перед использованием

Файл - os_tmr.c.

Вызов – только из *Задачи*.

Транслируется – если OS_TMR_EN>0.

Прототип: BOOLEAN OSTmrStart(OS_TMR *ptmr,
INT8U *perr);

Аргументы:

ptmr указатель на структуру типа OS_TMR, ассоциированную с *Таймером*.

perr адрес переменной в которую будет помещен код завершения, один из следующих:

OS_ERR_NONE	- если функция успешно запустила *Таймер*;
OS_ERR_TMR_INVALID	– если аргумент ptmr имеет значение NULL.
OS_ERR_TMR_INVALID_TYPE	– если аргумент ptmr указывает НЕ на *Таймер*;
OS_ERR_TMR_ISR	- если вызов произведен из обработчика прерывания;
OS_ERR_TMR_INACTIVE	- если ptmr указывает на неактивный *Таймер* (т.е. если последний еще не создан, либо уже удален).

Возвращаемое значение:

OS_TRUE	– если *Таймер* успешно запущен;
OS_FALSE	– если произошла ошибка (следует анализировать возвращаемое значение).

Замечания:
1) Следует всегда проверять возвращаемое значение, чтобы быть уверенным в результате.
2) Функцию нельзя вызывать из обработчика прерывания.
3) *Таймер* должен быть создан перед использованием.

Пример использования.

```
OS_TMR *CloseDoorTmr;
BOOLEAN status;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;

  for (;;) {
    status = OSTmrStart(CloseDoorTmr,
                        &err);
    if (err == OS_ERR_NONE) {
      /* Timer was started */
    }
  }
}
```

OSTmrStateGet()

C v2.83 ???

Описание – позволяет получить текущее состояние указанного *Таймера*, одно из следующих:

OS_TMR_STATE_UNUSED	данный *Таймер* не активен (еще не создан или уже удален);
OS_TMR_STATE_STOPPED	*Таймер* создан, но не запущен, либо остановлен;
OS_TMR_STATE_COMPLETED	*Таймер* в однократном режиме, и уже переполнился;
OS_TMR_STATE_RUNNING	*Таймер* работает.

Файл - os_tmr.c.

Вызов – только из *Задачи*.

Транслируется – если OS_TMR_EN>0.

Прототип INT8U OSTmrStateGet(OS_TMR *ptmr,
INT8U *perr);

Аргументы:

ptmr указатель на структуру типа OS_TMR, связанную с опрашиваемым *Таймером*;

perr адрес переменной в которую будет помещен код завершения, один из следующих:

OS_ERR_NONE	- если функция успешно запустила *Таймер*;
OS_ERR_TMR_INVALID	– если аргумент ptmr имеет значение NULL.
OS_ERR_TMR_INVALID_TYPE	– если аргумент ptmr указывает НЕ на *Таймер*;
OS_ERR_TMR_ISR	- если вызов произведен из обработчика прерывания;
OS_ERR_TMR_INACTIVE	- если ptmr указывает на неактивный *Таймер* (т.е. если последний еще не создан, либо уже удален).

Возвращаемое значение – состояние *Таймера*, следующие значения:

OS_TMR_STATE_UNUSED	данный *Таймер* не активен (еще не создан или уже удален);
OS_TMR_STATE_STOPPED	*Таймер* создан, но не запущен, либо остановлен;
OS_TMR_STATE_COMPLETED	*Таймер* в однократном режиме, и уже переполнился;
OS_TMR_STATE_RUNNING	*Таймер* работает.

Замечания:
1) Следует всегда проверять возвращаемое значение, чтобы быть уверенным в результате.
2) Функцию нельзя вызывать из обработчика прерывания.

Пример использования.

```
INT8U CloseDoorTmrState;
OS_TMR *CloseDoorTmr;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;

  for (;;) {
    CloseDoorTmrState = OSTmrStateGet(CloseDoorTmr, &err);
    if (err == OS_ERR_NONE) {
      /* Вызов завершился успешно */
    }
  }
}
```

OSTmrStop()

Функция введена в версии 2.81

Описание – ???

Файл - os_tmr.c.

Вызов – только из *Задачи*.

Транслируется – если OS_TMR_EN>0.

Прототип BOOLEAN OSTmrStop(OS_TMR * ptmr,
INT8U opt,
void * callback_arg,
INT8U * perr);

Аргументы:

ptmr указатель на структуру типа OS_TMR, связанную с опрашиваемым *Таймером*;

opt ???

callback_arg ???

perr адрес переменной в которую будет помещен код завершения, один из следующих:

OS_ERR_NONE если функция успешно запустила *Таймер*;OS_ERR_TMR_INVALID – если аргумент ptmr имеет значение NULL.
OS_ERR_TMR_INVALID_TYPE – если аргумент ptmr указывает НЕ на *Таймер*;
OS_ERR_TMR_ISR - если вызов произведен из обработчика прерывания (недопустимо);
OS_ERR_TMR_INVALID_OPT задано неверное значение параметра opt;
OS_ERR_TMR_STOPPED - *Таймер* уже остановлен (это не должно рассматриваться как ошибка);
OS_ERR_TMR_INACTIVE - если ptmr указывает на неактивный *Таймер* (т.е. если последний еще не создан, либо уже удален).
OS_ERR_TMR_NO_CALLBACK - если сделана попытка вызвать функцию callback, но она не была определена для указанного *Таймера*.

Возвращаемое значение:

OS_TRUE - если *Таймер* остановлен (или уже был остановлен);
OS_FALSE - если произошла ошибка (следует анализировать код завершения).

Замечания

1) Следует всегда проверять возвращаемое значение, чтобы быть уверенным в результате.

2) Функцию нельзя вызывать из обработчика прерывания.

3) Функция callback НЕ будет вызвана, если *Таймер* при вызове OSTmrStop() уже был остановлен

Пример использования.

```
OS_TMR *CloseDoorTmr;

void Task (void *p_arg) {
  INT8U err;
  (void)p_arg;
  for (;;) {
    OSTmrStop(CloseDoorTmr,
              OS_TMR_OPT_CALLBACK,
              (void *)0,
              &err);
```

```
    if (err == OS_ERR_NONE || err == OS_ERR_TMR_STOPPED) {
      /* Таймер остановлен */
      /* ... callback будет вызван только если Таймер работал */
    }
  }
}
```

OSVersion()

Описание – функция `OSVersion()` возвращает номер текущей версии `µC/OS-II`.

Файл - `OS_CORE.C`.

Вызов – из задачи или из обработчика прерывания

Транслируется – всегда.

Прототип `INT16U OSVersion(void);`

Аргументы - нет.

Возвращаемое значение – номер версии в формате `x.yy` умноженый на 100. Например, для версии 2.85 будет возвращено значение 285.

Замечания – нет.

Пример использования.

```
void TaskX (void *p_arg)  {
  INT16U os_version;
  for (;;) {
       ...
    os_version = OSVersion(); /* Obtain µC/OS-II's version */
       ...
  }
}
```

Сводная таблица свойств объектов синхронизации и коммуникации

Свойства объектов	Мьютексы	Семафоры	Почтовые ящики	Очереди сообщений	Флаги
Работают как двоичные семафоры	Да	Да	Да	Да	Да
Обеспечивают наследование приоритетов	Да	-	-	-	-
Позволяют задать тайм-аут	Да	Да	Да	Да	Да
Работают как счетные семафоры	-	Да	-	Да	-
Сигнализируют о событиях	-	Да	Да	Да	Да
Передают сообщения	-	-	Да	Да	-
Передают цепочку сообщений	-	-	-	Да	-
Поддерживают «логику» событий	-	-	-	-	Да
Поддерживают «широковещание»	-	-	Да	Да	-

Функции µC/OS-II и управляющие #define-константы.

Сервис	Установить не равной нулю для разрешения сервиса	Прочие константы
Разные сервисы		
OSEventNameGet()	OS_EVENT_NAME_SIZE (>1)	N/A
OSEventNameSet()	OS_EVENT_NAME_SIZE (>1)	N/A
OSInit()	N/A	OS_MAX_EVENTS OS_Q_EN and OS_MAX_QS OS_MEM_EN OS_TASK_IDLE_STK_SIZE OS_TASK_STAT_EN OS_TASK_STAT_STK_SIZE
OSSchedLock()	OS_SCHED_LOCK_EN	N/A
OSSchedUnlock()	OS_SCHED_LOCK_EN	N/A
OSStart()	N/A	N/A
OSStatInit()	OS_TASK_STAT_EN OS_TICKS_PER_SEC	OS_TASK_CREATE_EXT_EN
OSVersion()	N/A	N/A
Управление прерываниями		
OSIntEnter()	N/A	N/A
OSIntExit()	N/A	N/A
Флаги событий		
OSFlagAccept()	OS_FLAG_EN	OS_FLAG_ACCEPT_EN
OSFlagCreate()	OS_FLAG_EN	OS_MAX_FLAGS
OSFlagDel()	OS_FLAG_EN	OS_FLAG_DEL_EN
OSFlagNameGet()	OS_FLAG_EN	OS_FLAG_NAME_SIZE (>1)
OSFlagNameSet()	OS_FLAG_EN	OS_FLAG_NAME_SIZE (>1)
OSFlagPend()	OS_FLAG_EN	OS_FLAG_WAIT_CLR_EN
OSFlagPost()	OS_FLAG_EN	N/A
OSFlagQuery()	OS_FLAG_EN	OS_FLAG_QUERY_EN
Почтовые ящики		
OSMboxAccept()	OS_MBOX_EN	OS_MBOX_ACCEPT_EN
OSMboxCreate()	OS_MBOX_EN	OS_MAX_EVENTS
OSMboxDel()	OS_MBOX_EN	OS_MBOX_DEL_EN
OSMboxPend()	OS_MBOX_EN	N/A
OSMboxPost()	OS_MBOX_EN	OS_MBOX_POST_EN
OSMboxPostOpt()	OS_MBOX_EN	OS_MBOX_POST_OPT_EN
OSMboxQuery()	OS_MBOX_EN	OS_MBOX_QUERY_EN
Управление памятью		
OSMemCreate()	OS_MEM_EN	OS_MAX_MEM_PART
OSMemGet()	OS_MEM_EN	N/A
OSMemNameGet()	OS_MEM_EN	OS_MEM_NAME_SIZE (>1)
OSMemNameSet()	OS_MEM_EN	OS_MEM_NAME_SIZE (>1)
OSMemPut()	OS_MEM_EN	N/A
OSMemQuery()	OS_MEM_EN	OS_MEM_QUERY_EN
Мьютексы		
OSMutexAccept()	OS_MUTEX_EN	OS_MUTEX_ACCEPT_EN
OSMutexCreate()	OS_MUTEX_EN	OS_MAX_EVENTS
OSMutexDel()	OS_MUTEX_EN	OS_MUTEX_DEL_EN
OSMutexPend()	OS_MUTEX_EN	N/A
OSMutexPost()	OS_MUTEX_EN	N/A
OSMutexQuery()	OS_MUTEX_EN	OS_MUTEX_QUERY_EN
Очереди сообщений		
OSQAccept()	OS_Q_EN	OS_Q_ACCEPT_EN
OSQCreate()	OS_Q_EN	OS_MAX_EVENTS OS_MAX_QS
OSQDel()	OS_Q_EN	OS_Q_DEL_EN
OSQFlush()	OS_Q_EN	OS_Q_FLUSH_EN
OSQPend()	OS_Q_EN	N/A
OSQPost()	OS_Q_EN	OS_Q_POST_EN
OSQPostFront()	OS_Q_EN	OS_Q_POST_FRONT_EN
OSQPostOpt()	OS_Q_EN	OS_Q_POST_OPT_EN

Сервис	Установить не равной нулю для разрешения сервиса	Прочие константы
OSQQuery()	OS_Q_EN	OS_Q_QUERY_EN
Семафоры		
OSSemAccept()	OS_SEM_EN	OS_SEM_ACCEPT_EN
OSSemCreate()	OS_SEM_EN	OS_MAX_EVENTS
OSSemDel()	OS_SEM_EN	OS_SEM_DEL_EN
OSSemPend()	OS_SEM_EN	N/A
OSSemPost()	OS_SEM_EN	N/A
OSSemQuery()	OS_SEM_EN	OS_SEM_QUERY_EN
OSSemSet()	OS_SEM_EN	OS_SEM_SET_EN
Управление задачами		
OSTaskChangePrio()	OS_TASK_CHANGE_PRIO_EN	OS_LOWEST_PRIO
OSTaskCreate()	OS_TASK_CREATE_EN	OS_MAX_TASKS
OSTaskCreateExt()	OS_TASK_CREATE_EXT_EN	OS_MAX_TASKS OS_TASK_STK_CLR
OSTaskDel()	OS_TASK_DEL_EN	OS_MAX_TASKS
OSTaskDelReq()	OS_TASK_DEL_EN	OS_MAX_TASKS
OSTaskResume()	OS_TASK_SUSPEND_EN	OS_MAX_TASKS
OSTaskNameGet()	OS_TASK_NAME_SIZE (>1)	N/A
OSTaskNameSet()	OS_TASK_NAME_SIZE (>1)	N/A
OSTaskStkChk()	OS_TASK_CREATE_EXT_EN	OS_MAX_TASKS
OSTaskSuspend()	OS_TASK_SUSPEND_EN	OS_MAX_TASKS
OSTaskQuery()	OS_TASK_QUERY_EN	OS_MAX_TASKS
OS_TaskStatStkChk()	OS_TASK_STAT_STK_CHK_EN	N/A
Управление временем		
OSTimeDly()	N/A	N/A
OSTimeDlyHMSM()	OS_TIME_DLY_HMSM_EN	OS_TICKS_PER_SEC
OSTimeDlyResume()	OS_TIME_DLY_RESUME_EN	OS_MAX_TASKS
OSTimeGet()	OS_TIME_GET_SET_EN	N/A
OSTimeSet()	OS_TIME_GET_SET_EN	N/A
OSTimeTick()	N/A	N/A
Пользовательские функции		
OSTaskCreateHook()	OS_CPU_HOOKS_EN	N/A
OSTaskDelHook()	OS_CPU_HOOKS_EN	N/A
OSTaskStatHook()	OS_CPU_HOOKS_EN	N/A
OSTaskSwHook()	OS_CPU_HOOKS_EN	OS_TASK_SW_HOOK_EN
OSTimeTickHook()	OS_CPU_HOOKS_EN	OS_TIME_TICK_HOOK_EN