**Теория**

**ОСРВ**

Системы реального времени отличаются возможностью наступления серьёзных последствий в случае невыполнения ими в процессе работы не только логических, но и временных требований. Существует два типа систем реального времени: системы мягкого реального времени и системы жесткого реального времени. В системах мягкого реального времени задачи должны выполняться быстро, насколько это возможно, но конкретное время выполнения не задано. В системах жесткого реального времени задачи должны выполняться не только правильно, но и в срок. Большинство систем реального времени представляют собой комбинацию систем этих двух типов.

Ядро реального времени, также называемое операционной системой реального времени, или ОСРВ, позволяет проектировать и легко расширять приложения реального времени; дополнительные функции могут быть добавлены без существенных изменений в программном обеспечении. Применение ОСРВ позволяет упростить процесс проектирования за счёт разбиения кода приложения на отдельные задачи. Вытесняющая ОСРВ поддерживает все критичные ко времени события так быстро и эффективно, насколько это возможно. ОСРВ позволяет лучше использовать имеющиеся ресурсы, обеспечивая такие ценные сервисы как семафоры, почтовые ящики, очереди, задержки, тайм-ауты и другие.

**Многозадачность**

Многозадачность - это процесс переключения процессора между несколькими задачами. Многозадачная система похожа на систему, основанную на "суперпетле*" (т.е. беск. цикл + обработчики прерываний (почитайте в перевод User’s Manual))*, но с несколькими фонами. Многозадачность увеличивает использование процессора, а также обеспечивает возможность модульной конструкции приложения. Один из самых важных аспектов многозадачности состоит в том, что она позволяет программисту управлять сложностью, присущей приложениям реального времени. К тому же, при использовании многозадачности прикладные программы проще в написании и поддержке.

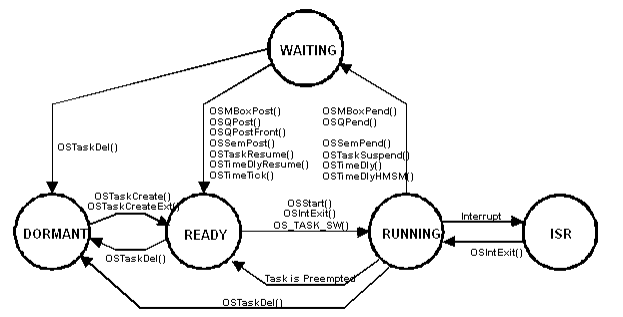
**Задача**

Задача, или поток, - это просто программа, полагающая, что только она одна выполняется на процессоре. Процесс проектирования приложения реального времени предполагает разделение всей задачи на части, каждая из которых ответственна за свою долю работы. Каждой задаче назначается приоритет, её собственный набор регистров процессора, и её область стека.

**Состояния задачи**

Каждая задача может находиться в одном из пяти состояний: сна, готовности, выполнения, ожидания события, или прерывания.

* Состояние сна соответствует задаче, которая находится в памяти, но не сделана доступной многозадачному ядру
* Задача находится в состоянии готовности, когда она может выполняться, но её приоритет меньше, чем у выполняющейся в данный момент задачи.
* Задача находится в состоянии выполнения, когда она управляет процессором.
* Задача находится в состоянии ожидания события, когда ей требуется появление события (такого как завершение операции ввода-вывода, освобождение разделяемого ресурса, появление тактирующего импульса, истечение времени, и т.п.).
* И, наконец, задача находится в состоянии прерывания, когда во время её выполнения возникло прерывание и процессор занят процедурой его обработки.



*(сейчас нас интересуют состояния ожидания, готовности и выполнения)*

**Переключение контекста (задач)**

Когда многозадачное ядро решает запустить следующую задачу, оно просто сохраняет контекст (содержимое регистров процессора) текущей задачи в её стеке - области хранения контекста задачи. Затем контекст следующей задачи восстанавливается из области его хранения, после чего продолжается выполнение кода задачи, чей контекст загружен. Такая последовательность действий называется переключением контекста, или переключением между задачами.

**Ядро**

Ядро - это часть многозадачной системы, ответственная за управление задачами (или за распределение времени процессора) и межзадачное взаимодействие. Основной сервис, предоставляемый ядром - это переключение контекста.

Ядро позволяет улучшить использование процессора, предоставляя такие обычно необходимые службы, как управление семафорами, почтовые ящики, очереди, временные задержки и т.п.

**Диспетчер задач**

Диспетчер задач - это часть ядра, ответственная за определение того, какая задача будет запущена следующей. Большинство ядер реального времени используют для этого систему приоритетов. Каждой задаче в зависимости от её важности назначается свой приоритет. Таким образом, в случае ядра реального времени с системой приоритетов управление процессором всегда будет получать готовая к выполнению задача с наивысшим приоритетом.

*(Далее речь о невытесняющих и вытесняющих ядрах, а также о реентерабельных функциях)*

МикроСи/ОС и большинство коммерческих ядер реального времени - это ядра вытесняющие.

**Приоритеты задач**

Каждой задаче назначается приоритет, тем высший, чем задача важнее. В большинстве реализаций ядер ОС РВ прикладной программист сам решает, как назначать приоритеты.

МикроСи/ОС-II позволяет динамически изменять приоритеты задач.

**Немножко о прерываниях (возврат из прерываний)**

Обычно, в конце подпрограммы обработки прерывания необходимо вызывать специальную функцию выхода из обработчика прерывания. В микроСи/ОС-II эта функция называется **OSIntExit().** Эта функция позволяет ядру следить за вложенностью прерываний. В случае возврата из прерывания низшего уровня к выполнению кода уровня задач, сервисы ядра определяют задачу наивысшего приоритета, находящуюся в состоянии готовности. Если в результате выполнения ISR некая высокоприоритетная задача станет готовой к исполнению, она получит управление процессором.

*(для таймерного прерывания это полностью справедливо)*

**Таймерное прерывание**

Таймерное прерывание, или прерывание от таймера - это специальное прерывание, регулярное срабатывание которого обеспечивается аппаратно. Таймерное прерывание сродни биению сердца системы (spoooooooky). Таймерное прерывание позволяет ядру задерживать выполнение задач на целое количество периодов таймера и обеспечивать выдержку тайм-аутов задачам, ожидающим возникновения событий. Чем выше темп поступления прерываний от таймера, тем больше затраты процессорного времени в системе (и тем выше точность выдержки интервалов времени - прим. перев.).

Все ядра позволяют приостанавливать выполнение задач на определённое количество периодов таймера. Разрешающая способность по времени, в данном случае, составляет один период таймера, что, конечно, не означает и точность выдержки в один период.

**Устройство Ядра**

**OS\_ENTER\_CRITICAL() и OS\_EXIT\_CRITICAL()**

Как и все ядра реального времени, микроСи/ОС-II запрещает прерывания на время выполнения критических секций кода. Чтобы скрыть конкретный метод, избранный производителем компилятора, в микроСи/ОС-II для запрещения и разрешения прерываний определены два макроса **- OS\_ENTER\_CRITICAL()** и **OS\_EXIT\_CRITICAL()**, соответственно.

**Задачи**

Задача обычно представляет собой функцию, содержащую бесконечный цикл. Задача выглядит так же, как и любая другая функция на Си, имеющая возвращаемое значение и аргументы, однако она никогда не возвращает значения. Поэтому тип возвращаемого значения всегда должен быть объявлен как пустой тип, то есть **void**.

Кроме того, задача может прекратить своё выполнение. При этом сам код задачи не пропадает, но микроСи/ОС-II больше не будет принимать данную задачу во внимание, поэтому её код больше не получит управление (**OSTaskDel()**).

Аргумент **void\* pdata** передаётся задаче, когда она начинает выполнение. Обратите внимание, что он представляет собой указатель типа **void**. Это позволяет приложению передавать задаче практически любой тип данных. Такой указатель является универсальным средством, позволяющим передать задаче, например, адрес переменной, структуру, или даже адрес функции, если это необходимо. Поэтому возможно создание нескольких одинаковых задач, использующих одну и ту же функцию, или даже тело задачи.

Каждой задаче должен быть присвоен уникальный уровень приоритета, причём чем меньше порядковый номер уровня, тем выше приоритет задачи. МикроСи/ОС-II всегда выполняет готовую задачу наивысшего приоритета. В текущей версии микроСи/ОС-II номер уровня приоритета служит также идентификатором задачи.

Для того, чтобы микроСи/ОС-II могла управлять задачей, задачу сначала необходимо "создать". Вы создаёте задачу, передавая её адрес и аргументы одной из двух функций: OSTaskCreate(), *(есть еще одна, но её опускаем).*

**Состояния задачи**

Механизм многозадачности начинает действовать после вызова **OSStart(). OSStart()** запускает на выполнение задачу с наивысшим приоритетом, находящуюся в состоянии готовности. При этом данная задача переводится в состояние выполнения. В любой момент времени может выполняться только одна задача. Задача не переводится из состояния готовности в состояние выполнения, пока все задачи с более высокими уровнями приоритета либо не перейдут в состояние ожидания, либо не будут прекращены.

Выполняющаяся задача может приостановить своё выполнение на определённое количество времени, вызывая **OSTimeDly()** или *(есть еще одна, но её опускаем).* При этом задача переводится в состояние ожидания истечения некоторого промежутка времени, а управление незамедлительно передаётся следующей задаче с наивысшим приоритетом, находящейся в состоянии готовности. По истечении требуемого интервала времени задержанная задача переводится в состояние готовности процедурой **OSTimeTick()**.

Задаче также может быть необходимо дождаться появления некоторого события, что обеспечивается при помощи таких сервисов, как **OSSemPend()**, **OSMboxPend()** или **OSQPend().** При этом задача переводится в состояние ожидания появления события, а управление незамедлительно передаётся следующей задаче с наивысшим приоритетом, находящейся в состоянии готовности. При появлении события задача переводится в состояние готовности. Сигнализирование о появлении события производится другой задачей или ISR.

Когда все имеющиеся задачи находятся либо в состоянии ожидания события, либо в состоянии ожидания истечения временного интервала, то во время простоя микроСи/ОС-II выполняет "пустую" задачу, **OSTaskIdle().**

**Блоки управления задачами (OS\_TCB)**

Когда вы создаёте задачу, ей сопоставляется структура данных, называемая блоком управления задачей, **OS\_TCB**, которая используется для хранения состояния задачи, пока она вытеснена. Когда задача вновь получает управление процессором, блок контроля позволяет продолжить выполнение задачи точно с того места, на котором выполнение прекратилось.

Поля **OS\_TCB** инициализируются при создании задачи.

Поля OS\_TCB (*на самом деле, не все*):

* **OSTCBStkPtr** содержит указатель на текущую вершину стека задачи.
* **OSTCBStkBottom** содержит указатель на основание стека задачи.

**OSTCBStkSize** содержит размер стека в элементах, а не в байтах.

* **OSTCBNext** и **OSTCBPrev** используются для связывания блоков управления задачей **OS\_TCB**. Цепь из блоков **OS\_TCB** используется **OSTimeTick()** для обновления полей **OSTCBDly** каждой задачи. Связывание блоков **OS\_TCB** производится при создании задачи, а при удалении задачи блок **OS\_TCB** удаляется из этого списка.
* **OSTCBDly** используется при необходимости задержать выполнение задачи на определённое число периодов системного таймера, или если задаче необходимо дожидаться возникновения некоторого события определённое время (тайм-аут). В последнем случае это поле содержит число тиков таймера, которое задаче позволено ожидать события. Когда это число равно нулю, задача либо не должна задерживаться вообще, либо необходимо дожидаться возникновения события неопределённое время (т.е. без тайм-аута).
* **OSTCBStat** содержит состояние задачи. Когда содержимое **OSTCBStat** равно нулю, задача готова к выполнению. **OSTCBStat** также может принимать другие значения, описываемые в **uCOS\_II.H**.
* **OSTCBPrio** содержит приоритет задачи. Задача имеет тем больший приоритет, чем меньше величина **OSTCBPrio**.
* **OSTCBX**, **OSTCBY**, **OSTCBBitX** и **OSTCBBitY** используются для ускорения процессов перевода задачи в состояния готовности и ожидания события и позволяют избегать вычисления их величин во время исполнения задачи.

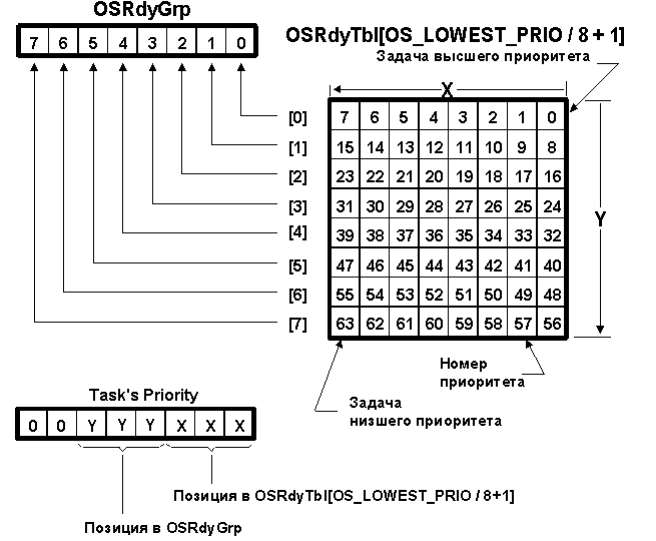
Максимальное количество задач (**OS\_MAX\_TASKS**), которые могут быть в приложении, определяется в файле **OS\_CFG.H**, и определяет количество блоков **OS\_TCB**, выделяемых микроСи/ОС-II вашему приложению.

Все **OS\_TCB** помещаются в **OSTCBTbl[]**. При инициализации микроСи/ОС-II все блоки **OS\_TCB** находятся в связанном списке свободных **OS\_TCB**. При создании задачи ей передаётся блок **OS\_TCB**, на который указывает **OSTCBFreeList**, а **OSTCBFreeList** перенаправляется на следующий в списке блок. При удалении задачи её блок **OS\_TCB** возвращается в список свободных блоков **OS\_TCB**.

**Лист готовности**

Каждая задача, находящаяся в состоянии готовности к выполнению, помещается в лист готовности, состоящий из двух переменных - **OSRdyGrp** и **OSRdyTbl[].** В **OSRdyGrp** задачи сгруппированы по восемь, и каждый бит этой переменной обозначает готовность к выполнению любой задачи из соответствующей группы. Готовая к выполнению задача устанавливает также соответствующий ей бит в таблице готовности, **OSRdyTbl[]**, размер которой определяется **OS\_LOWEST\_PRIO**.

Чтобы определить задачу, которая будет выполняться следующей, то есть задачу с наивысшим приоритетом, диспетчер определяет задачу с наименьшим номером приоритета, установившую свой бит готовности в **OSRdyTbl[]**.



Для помещения задачи в лист готовности используется следующий фрагмент кода:

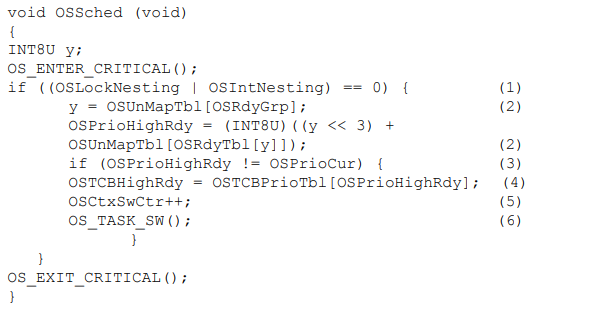


где **prio** - приоритет задачи.

**Диспетчеризация задач**

МикроСи/ОС-II всегда выполняет задачу с наивысшим приоритетом. Определение того, какая задача имеет высший приоритет, и, соответственно, какая задача будет выполняться, производится диспетчером. Диспетчеризация уровня задач осуществляется OSSched(). Диспетчеризация уровня прерываний осуществляется другой функцией OSIntExit().

Время диспетчеризации задач в микроСи/ОС-II является величиной постоянной, безотносительной количеству задач, запущенных в приложении. **OSSched()** прекращает выполнение, если запущена из процедуры обработки прерывания ( т.е. если **OSIntNesting** > 0) или если диспетчеризация была отключена хотя-бы одним вызовом **OSSchedLock()** (т.е. **OSLockNesting** > 0). Если **OSSched()** была вызвана не из ISR, а диспетчеризация разрешена, **OSSched()** определяет задачу с наивысшим приоритетом, готовую к выполнению, то есть имеющую установленный бит готовности в **OSRdyTbl[]**. Когда высокоприоритетная задача найдена, **OSSched()** проверяет, не является ли она текущей задачей. Это делается для того, чтобы избежать ненужного переключения контекста. Обратите внимание, что микроСи использовала для этой проверки сравнение **OSTCBHighRdy** и **OSTCBCur**. На 8- и некоторых 16-разрядных микропроцессорах эта операция выполнялась относительно медленно, так как производилось сравнение указателей, а не байтов, как это делается в микроСи/ОС-II. Также, нет смысла искать **OSTCBHighRdy** в **OSTCBPrioTbl[]**, пока не нужно переключать контекст. Сравнение байтов вместо указателей, а также поиск **OSTCBHighRdy** только при необходимости должно сделать микроСи/ОС-II быстрее микроСи/ОС на 8- и некоторых 16-разрядных процессорах.



***(код в os\_core.c)***

***(у нас OSSched() вызывется в OSTimeDly(), например)***

Для переключения контекста необходимо, чтобы **OSTCBHighRdy** указывала на **OS\_TCB** задачи наивысшего приоритета, что достигается использованием **OSTCBHighRdy** в качестве индекса **OSTCBPrioTbl[]**

Переключение контекста состоит из сохранения регистров процессора на стеке приостанавливаемой задачи и восстановления регистров из стека задачи, запускаемой на выполнение. В микроСи/ОС-II, область стека готовой к выполнению задачи выглядит так, как если бы только что произошло прерывание, и регистры процессора были помещены в стек. Другими словами, всё, что микроСи/ОС-II должна сделать для запуска готовой задачи, это восстановить регистры процессора из стековой области данной задачи, и выполнить инструкцию возврата из прерывания. Поэтому **OS\_TASK\_SW()** должна быть реализована так, как будто она моделирует прерывание. Большинство процессоров поддерживает это либо посредством программных прерываний, либо при помощи специальной инструкции **TRAP** ("ловушка"). Обработчик этого прерывания, либо обработчик данной ловушки (также называемый обработчиком исключения) ДОЛЖЕН перенаправляться на ассемблерную функцию **OSCtxSw()**, которая ожидает, что **OSTCBHighRdy** указывает на **OS\_TCB** запускаемой задачи, а **OSTCBCur** указывает на **OS\_TCB** приостанавливаемой задачи. Дополнительные подробности см. в Главе 8, Портирование микроСи/ОС-II.

**Задача бездействтия**

МикроСи/ОС-II всегда содержит в своём составе так называемую "пустую" задачу, которая запускается в те моменты, когда нет других задач, готовых к выполнению, то есть когда процессор простаивает. "Пустая" задача, **OSTaskIdle()**, всегда имеет самый низкий уровень приоритета, т.е. **OS\_LOWEST\_PRIO**.

**OSTaskIdle()** создается в **OSInit()**.

**Таймерное прерывание**

Таймерное прерывание должно быть разрешено после запуска многозадачности, то есть после вызова **OSStart()**. Другими словами, нужно инициализировать и запускать таймерное прерывание в первой же задаче, следующей за вызовом **OSStart().**

Таймерное прерывание микроСи/ОС-II обрабатывается подпрограммой его обслуживания при помощи вызова такой системной службы, как **OSTimeTick().**

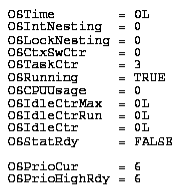
Большая часть работы **OSTimeTick()** - это, в основном, декрементирование полей **OSTCBDly** каждого TCB, если они ненулевые. **OSTimeTick()** следует по цепи **OS\_TCB**, начиная с **OSTCBList**, и заканчивая "пустой" задачей. Когда поле **OSTimeDly** блока **OS\_TCB** задачи декрементируется до нуля, задача делается готовой к выполнению.

Время выполнения **OSTimeTick()** прямо пропорционально числу задач, имеющихся в приложении.

**Запуск uC/OS-II**

Будучи вызвана, **OSStart()** находит **OS\_TCB** задачи наивысшего приоритета (посредством просмотра листа готовности). Затем **OSStart()** вызывает **OSStartHighRdy()**. **OSStartHighRdy()** восстанавливает регистры процессора, выталкивая их из стека задачи, а затем выполняет инструкцию возврата из прерывания, чем заставляет процессор перейти к выполнению кода. Обратите внимание, что выполнение **OSStartHighRdy()** не возвращается в **OSStart()**.

**OSTaskCtr**

****

**OSTaskSuspend**

Иногда бывает нужно приостановить выполнение задачи явным образом. Это можно сделать при помощи вызова функции **OSTaskSuspend()**. Приостановленная задача может быть вновь запущена только при помощи вызова функции **OSTaskResume()**. Приостановка задачи является аддитивной. Это означает, что если приостановленная задача ожидает ещё и истечения промежутка времени, то для запуска задачи приостановка должна быть снята и промежуток времени должен истечь. Задача может приостановить другую задачу или саму себя.

В обоих этих случаях необходимо будет вызвать диспетчер, поэтому была создана локальная переменная **self**, которая в соответствующее время будет проверяться. Если же мы не пытаемся остановить текущую задачу, вызывать диспетчер нет необходимости, так как в этом случае текущая задача приостанавливает задачу более низкого приоритета.

Затем проверяется, существует ли приостанавливаемая задача. Если она существует, она извлекается из листа готовности.

В блоке **OS\_TCB** задачи устанавливается флаг **OS\_STAT\_SUSPEND**, чтобы отразить её состояние.

**OSTaskResume()** продолжает выпонение задачи.

**Задержка выполнения задачи, функция OSTimeDly()**

МикроСи/ОС-II содержит сервис, позволяющий вызывающей его задаче задерживать своё выполнение на определённое количество периодов таймера, или тиков. Обеспечивающая этот сервис функция называется **OSTimeDly()**. Вызов этой функции приводит к переключению контекста и заставляет микроСи/ОС-II выполнять следующую задачу с высшего приоритета, готовую к выполнению. Задача, вызвавшая функцию **OSTimeDly()**, становится готовой к выполнению по мере истечения заданного интервала времени, или когда другая задача отменит задержку посредством вызова функции **OSTimeDlyResume()**. Обратите внимание, что эта задача выполняется только тогда, когда она имеет высший приоритет.

* Если задано значение 0, вы показываете, что не хотите задерживать задачу вообще, при этом функция сразу возвращает управление обратно вызвавшей её задаче
* Ненулевая величина переменной **ticks** приводит к тому, что функция **OSTimeDly()** удаляет текущую задачу из листа готовности
* Затем значение переменной **ticks** *(аргумент* ***OSTimeDly()****)*сохраняется в блоке **OS\_TCB** текущей задачи, где при помощи функции **OSTimeTick()** оно будет уменьшаться на единицу каждый период таймера. Обратите внимание, что задача не помещается ни в какой лист ожидания. Ненулевого значения поля **OSTCBDly** достаточно, чтобы функция **OSTimeTick()** считала задачу задержанной.
* Наконец, поскольку задача более не готова к выполнению, вызывается диспетчер для переключения на следующую задачу высшего приоритета, готовую к выполнению.

**OSTimeDlyHMSM()** позволяет задавать задержку в секундах, нежели чем в периодах системного таймера.