**Концепции систем реального времени**

Системы реального времени характеризуются тяжелыми последствиями, которые возникают, если не соблюдаются логические свойства, а также свойства правильности синхронизации системы. Существует два типа систем реального времени: SOFT и HARD. В системе SOFT в режиме реального времени задачи выполняются системой как можно быстрее, но задачи не должны заканчиваться к определенному времени. В HARD системах реального времени задачи должны выполняться не только правильно, но и вовремя. Большинство систем реального времени имеют сочетание требований SOFT и HARD. Приложения реального времени охватывают широкий диапазон, но большинство систем реального времени встроены. Это означает, что компьютер встроен в систему и не рассматривается пользователем как компьютер. В следующем списке приведены несколько примеров встраиваемых систем.

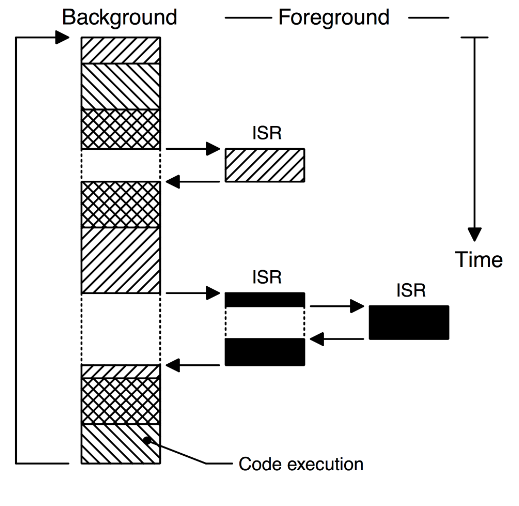
|  |  |
| --- | --- |
| **Управление технологическим процессом**  Пищевая промышленность  Химические заводы  **Автомобильный**  Органы управления двигателем  Антиблокировочные тормозные системы  **Автоматизация делопроизводства**  Факсы  Копировальные устройства  **Периферийные устройства компьютера**  Принтеры  Терминалы  Сканеры  Модемы | **Коммуникация**  Выключатели  Маршрутизаторы  **Роботы**  **Космос**  Системы управления полетами  Системы вооружения Управление реактивными двигателями  **Внутренний**  Микроволновые печи  Посудомоечные машины  Стиральные машины  Термостаты |

Программные приложения реального времени, как правило, более сложны в разработке, чем приложения не реального времени. В этой главе описываются концепции реального времени.

**Системы переднего/заднего плана**

Небольшие системы низкой сложности обычно проектируются, как показано на рис. 2.1. Эти системы называются основной системой переднего/заднего плана или суперциклами. Приложение состоит из бесконечного цикла, который вызывает модули (т.е. функции) для выполнения требуемых операций (фон). Подпрограммы обслуживания прерываний (ISRs) обрабатывают асинхронные события (передний план). Передний план также называется уровнем прерывания; фон называется уровнем задачи. Критически важные операции должны выполняться ISR для обеспечения их своевременного выполнения. Из-за этого ISR имеют тенденцию занимать больше времени, чем они должны. Кроме того, информация для модуля системы , предоставляемая ISR, не обрабатывается до тех пор, пока программа системы не выполнит свою очередь. Это называется ответом на уровне задачи. Время отклика на уровне задачи в наихудшем случае зависит от продолжительности выполнения фонового цикла. Так как время выполнения типичного кода не является постоянным, время прохождения через часть цикла для sµCcessive является недетерминированным. Кроме того, если выполняется изменение кода, это влияет на синхронизацию цикла.

**Рисунок 2.1 Системы** **переднего/заднего плана**



Большинство приложений на основе микроконтроллеров большого объема (например, микроволновые печи, телефоны, игрушки и т.д.) разработаны как системы переднего/заднего плана. Кроме того, в приложениях на основе микроконтроллеров может быть лучше (с точки зрения энергопотребления) остановить процессор и выполнить всю обработку в ISR. Тем не менее, вы также можете остановить работу процессора, когда µC/OS-II не имеет каких либо задач для выполнения.

**Код критических секций**

Критическим разделом кода, также называемым критической областью, является код, который должен рассматриваться неделимо. Как только раздел кода начинает выполняться, он не должен прерываться. Для того чтобы это обеспечить,прерывания обычно отключаются перед выполнением критического кода и активируются после его завершения (см. также раздел 2.03, Общий ресурс).

**Ресурс**

Ресурс - это любая сущность, используемая задачей. Таким образом, ресурсом может быть устройство ввода-вывода, такое как принтер, клавиатура или дисплей, или переменная, структура или массив.

**Общий ресурс**

Общий ресурс - это ресурс, который может использоваться несколькими задачами. Каждая задача должна получить эксклюзивный доступ к общему ресурсу для предотвращения повреждения данных. Это называется взаимным исключением, и методы обеспечения взаимного исключения обсуждаются в разделе 2.18, Взаимное исключение.

**Многозадачность**

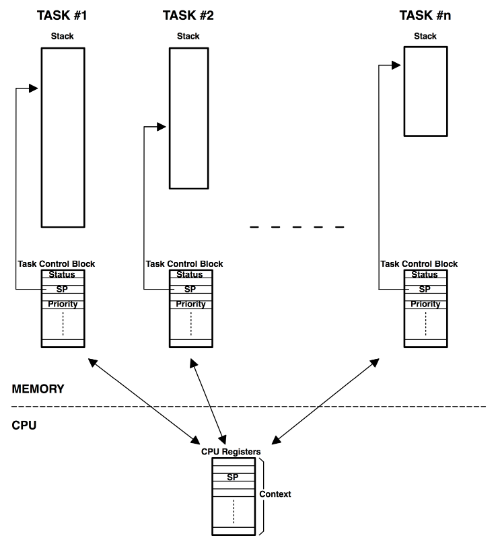
Многозадачность - это процесс планирования и переключения ЦП (центрального процессора) между несколькими задачами; один ЦП переключает внимание между несколькими последовательными задачами. Многозадачность подобна переднему/заднему плану с несколькими фонами. Многозадачность максимально увеличивает загрузку ЦП, а также обеспечивает модульную конструкцию приложений. Одним из наиболее важных аспектов многозадачности является то, что она позволяет программисту приложений управлять сложностью, присущей приложениям реального времени. Прикладные программы, как правило, легче проектировать и обслуживать, если используется многозадачность.

**Задача**

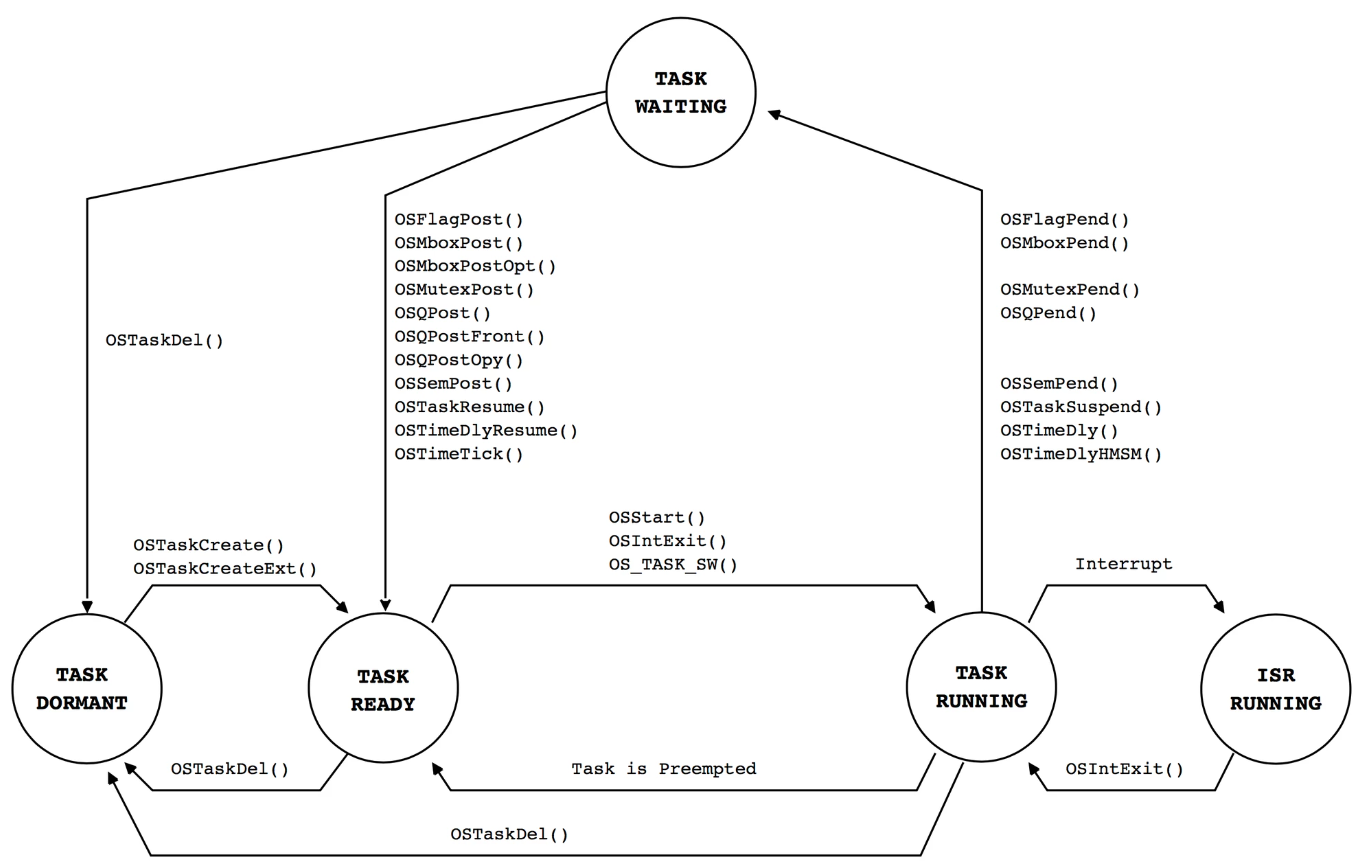
Задача, также называемая потоком, представляет собой простую программу, которая считает, что у нее есть процессор. Процесс разработки приложения в реальном времени включает разделение выполняемой работы на задачи, ответственные за часть проблемы. Каждой задаче присваивается приоритет, собственный набор регистров CPU и своя область стека (как показано на рис. 2.2).

Каждая задача обычно представляет собой бесконечный цикл, который может находиться в любом из пяти состояний: DORMANT, READY, RUNNING, WAITING  (для события) или ISR  (прерывание) (рис. 2.3). Состояние DORMANT  соответствует задаче, находящейся в памяти, но недоступной для многозадачного ядра. Задача READY , когда она может выполняться, но ее приоритет меньше, чем текущая выполняемая задача. Задача RUNNING , когда она управляет ЦП. Задача находится в состоянии WAITING , когда требуется возникновение события (ожидание завершения операции ввода-вывода, доступность совместно используемого ресурса, возникающий импульс синхронизации, время окончания операции и т.д.). Наконец, задача находится в состоянии ISR , когда произошло прерывание и CPU находится в процессе обслуживания прерывания. На рисунке 2.3 также показаны функции, обеспечиваемые uС/ОС-II для перевода задачи из одного состояния в другое.

**Рисунок 2.2 Множество задач**



**Рисунок 2.3 Состояния задач**



**Переключение контекста (переключение задачи)**

Когда многозадачное ядро решает управлять другой задачей, оно просто сохраняет контекст текущей задачи (регистры центрального процессора) в области хранения контекста текущей задачи — его стек (рисунок 2.2). Как только эта операция выполняется, контекст новой задачи восстанавливается из его области хранения, тогда возобновляется выполение кода новой задачи. Этот процесс называют переключением контекста или переключением задачи. Переключение контекста добавляет надстройку к приложению. Чем больше регистров центральный процессор имеет, тем больше надстройка. Время, требуемое для выполнения переключения контекста, определено тем, сколько регистров должно быть сохранено и восстановлено центральным процессором. Производительность ядра в реальном времени не стоит оценивать по тому, сколько переключений контекста ядро способно выполнить в секунду.

**Ядро**

Ядро - часть многозадачной системы, ответственной за управление задачами (т.е. для управления временем центрального процессора) и связь между задачами. Фундаментальный серис, предоставляемый ядром, является переключением контекста. Использование ядра в реальном времени обычно упрощает дизайн систем, позволяя приложению быть разделенным на несколько задач, управляемых ядром.

Ядро добавляет надстройку к Вашей системе, потому что сервисы, предоставленные ядром, требуют времени выполнения. Величина надстройки зависит от того, как часто Вы вызываете эти сервисы. В хорошо разработанном приложении ядро израсходует между 2 и 5% времени центрального процессора. Поскольку ядро - программное обеспечение, которое добавлено к Вашему приложению, оно требует дополнительного ROM (кодовое пространство) и дополнительная RAM для структур данных ядра, к тому же каждая задача требует своего собственного пространства стека, у которого есть тенденция съесть RAM быстро.

Однокристальные микроконтроллеры обычно не в состоянии управлять ядром в реальном времени, потому что у них есть очень мало RAM. Ядро позволяет Вам лучше использовать свой центральный процессор, предоставляя Вам обязательные службы, такие как управление семафором, почтовые ящики, очереди, временные задержки, и т.д. Как только Вы проектируете систему, используя ядро в реальном времени, Вы не захотите возвращаться к системе переднего плана/фона.

**Планировщик**

Планировщик, также названный диспетчером, является частью ядра, ответственного за определение, какая задача будет управляться следующей. Большинство ядер реального времени являются приоритетными. Каждой задаче назначают приоритет на основе ее важности. Приоритет для каждой задачи зависит от конкретного приложения. В основанном на приоритете ядре контроль центрального процессора всегда дается к самой высокой приоритетной задаче, готовой выполнятся. То, когда самая высокая приоритетная задача получает центральный процессор, однако, определено типом используемого ядра. Есть два типа основанных на приоритете ядер: неприоритетные и приоритетные.

**Неприоритетное ядро**

Неприоритетные ядра требуют, чтобы каждая задача сделала что-то, чтобы явно передать контроль центральному процессору. Чтобы поддержать иллюзию параллелизма, этот процесс должен часто выполнятся. Неупреждающее планирование также называют совместной многозадачностью; задачи сотрудничают друг с другом, чтобы разделить центральный процессор. Асинхронные события все еще обрабатываются прерываниями ISRs. ISR  может сделать более высокую приоритетную задачу готовой выполняться, но ISR  всегда возвращается к прерванной задаче. Новая более высокая приоритетная задача получит контроль над центральным процессором только, когда текущая задача освободит центральный процессор.

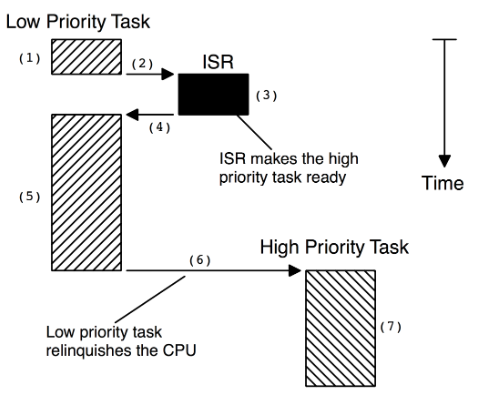
Одно из преимуществ неприоритетного ядра - то, что задержка выполнения прерывания типично низкая (см. более позднее обсуждение прерываний). На уровне задачи неприоритетные ядра могут также использовать функции non-reentrant (обсудится позже). Функции Non-reentrant могут использоваться каждой задачей без страха перед повреждением другой задачи. Это вызвано тем, что каждая задача может выполнятся до завершения, прежде чем это освободит центральный процессор. Однако функциям non-reentrant нельзя позволять освободить контроль центрального процессора.

Отклик задачи, используя неприоритетное ядро может быть намного ниже, чем с системами переднего плана/фона, потому что отклик уровня задачи теперь зависит от времени самой долгой задачи.

Другое преимущество неприоритетных ядер - меньшая потребность охранять совместно используемые данные с помощью семафоров. Каждая задача владеет центральным процессором, и Вы не должны бояться, что задача будет выгружена. Это не абсолютное правило, и в некоторых случаях, семафоры должны все еще использоваться. Общие устройства ввода/вывода могут все еще потребовать использования взаимных семафоров исключения; например, задаче, возможно, все еще понадобился бы исключительный доступ к принтеру.

Профиль выполнения неприоритетного ядра показан на рисунке 2.4 и описывается ниже.

**Рисунок 2.4 неприоритетное ядро**



(1) Задача выполняется, но прерывается.

(2) Если прерывания активизированы, векторы CPU  (переходят) к ISR.

(3) ISR обрабатывает событие F2.4 (3) и делает задачу с более высоким приоритетом готовой к выполнению.

(4) После завершения ISR выполняется команда Return From Interrupt, и CPU  возвращается к прерванной задаче.

(5) Код задачи возобновляется в команде, следующей за прерванной командой.

(6) Когда код задачи завершается, он вызывает службу, предоставляемую ядром, чтобы передать CPU  другой задаче.

(7) Ядро видит, что задача с более высоким приоритетом была готова к выполнению (оно не обязательно знает, что она была от ISR , и не заботится о ней), и, таким образом, ядро выполняет переключение контекста, чтобы оно могло запустить (то есть выполнить) задачу с более высоким приоритетом для обработки события, сообщенного ISR .

Самый главный недостаток неупреждающего ядра - отзывчивость. Задача с более высоким приоритетом, которая была готова к запуску, может долго ждать запуска, поскольку текущая задача должна отказаться от ЦП, когда она будет готова к этому. Как и в случае фонового выполнения в системах суперцикла, время отклика на уровне задачи в необеспеченном ядре является недетерминированным; вы никогда не знаете, когда задача с наивысшим приоритетом получит контроль над CPU. Ваше приложение должно отказаться от управления ЦП.

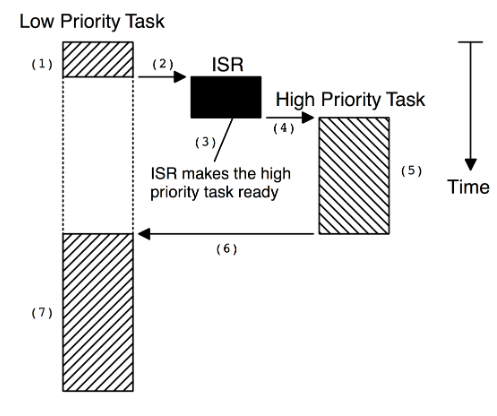
Подводя итог, можно сказать, что неприоритетное ядро позволяет выполнять каждую задачу до тех пор, пока оно добровольно не откажется от управления ЦП. Прерывание вытесняет задачу. По завершении ISR ISR возвращается к прерванной задаче. Реакция на уровне задач намного лучше, чем при использовании системы с суперциклом, но все еще является недетерминированной. Очень мало коммерческих ядер являются неприоритетными.

**Упреждающее ядро**

Упреждающее ядро используется, когда важна оперативность системы. В связи с этим, uC/OS-II и большинство коммерческих ядер реального времени являются упреждающими. Задача с наивысшим приоритетом, готовая к выполнению, всегда находится под управлением ЦП. Когда задача делает задачу с более высоким приоритетом готовой к выполнению, текущая задача вытесняется (приостанавливается) и задача с более высоким приоритетом немедленно получает управление CPU. Если ISR обеспечивает готовность задачи с более высоким приоритетом, то после завершения ISR прерванная задача приостанавливается и возобновляется новая задача с более высоким приоритетом.

Профиль выполнения приоритетного ядра показан на рис. 2.5 и описан ниже.

**Рисунок - Рисунок 2.5 Приоритетное ядро**



(1) Задача выполняется, но прерывается.

(2) Если прерывания активизированы, векторы CPU  (переходят) к ISR.

(3) ISR обрабатывает событие и делает задачу с более высоким приоритетом готовой к выполнению. По завершении ISR вызывается услуга, предоставляемая ядром (т.е. вызывается функция, предоставляемая ядром).

(4) & (5) Эта функция знает, что более важная задача была готова к выполнению, и, таким образом, вместо возврата к прерванной задаче ядро будет выполнять переключение контекста и выполнять код более важной задачи. Когда выполняется более важная задача, вызывается другая функция, предоставляемая ядром, чтобы перевести задачу в спящий режим в ожидании события (т.е. IS ISR R).

(6) (7) Затем ядро «видит», что задача с более низким приоритетом должна выполняться, и для возобновления выполнения прерванной задачи выполняется другое переключение контекста.

С приоритетным ядром выполнение задачи с наивысшим приоритетом является детерминированным; вы можете определить, когда она получит управление ЦП. Таким образом, время отклика на уровне задачи минимизируется за счет использования приоритетного ядра.

Код приложения, использующий приоритетное ядро, не должен использовать повторно применяемые функции, если только исключительный доступ к этим функциям не обеспечивается посредством использования семафоров взаимного исключения, поскольку как низко-, так и высокоприоритетная задача может использовать общую функцию. Повреждение данных может произойти, если задача с более высоким приоритетом вытесняет задачу с более низким приоритетом, которая использует функцию.

Таким образом, упреждающее ядро всегда выполняет задачу наивысшего приоритета, готовую к выполнению. Прерывание вытесняет задачу. После завершения ISR ядро возобновляет выполнение задачи с наивысшим приоритетом, готовой к выполнению (а не прерванной задачи). Реакция на уровне задач является оптимальной и детерминированной uC/OS-II - это упреждающее ядро.