Лекция 3 Особенности построения ОСРВ

План:
1 Системы исполнения и системы разработки в операционных системах реального времени
2 Ядро
3 Методы планирования ОС

1 Системы исполнения и системы разработки в операционных системах реального времени.

Одно из коренных внешних отличий систем реального времени от систем общего назначения - четкое разграничение систем разработки и систем исполнения.

Система исполнения операционных системах реального времени - это набор инструментов (ядро, драйверы, исполняемые модули), обеспечивающих функционирование приложения реального времени. Большинство современных ведущих операционных систем реального времени поддерживают целый спектр аппаратных архитектур, на которых работают системы исполнения (Intel, Motorola, RISC, MIPS, PowerPC, и другие). Это объясняется тем, что набор аппаратных средств - часть комплекса реального времени и аппаратура должна быть также адекватна решаемой задаче, поэтому ведущие операционные системы реального времени перекрывают целый ряд наиболее популярных архитектур, чтобы удовлетворить самым разным требованиям по части аппаратуры. Система исполнения операционных системах реального времени и компьютер, на котором она исполняется, называют "целевой" (target) системой.

Требования, предъявляемые к среде исполнения систем реального времени, следующие:

- небольшая память системы - для возможности ее встраивания;
- система должна быть полностью резидентна в памяти, чтобы избежать замещения страниц памяти или подкачки;
- система должна быть многозадачной - для обеспечения максимально эффективного использования всех ресурсов системы;
- ядро с приоритетом на обслуживание прерывания. Приоритет на прерывание означает, что готовый к запуску процесс, обладающий некоторым приоритетом, обязательно имеет преимущество в очереди по отношению к процессу с более низким приоритетом, быстро заменяет последний и поступает на выполнение. Ядро заканчивает любую сервисную работу, как только поступает задача с высшим приоритетом. Это гарантирует предсказуемость системы;
- диспетчер с приоритетом - дает возможность разработчику прикладной программы присвоить каждому загрузочному модулю приоритет, неподвластный системе. Присвоение приоритетов используется для определения очередности запуска программ, готовых к исполнению. Альтернативным такому типу диспетчеризации является диспетчеризация типа "карусель", при которой каждой готовой к продолжению программе дается равный шанс запуска. При использовании этого метода нет контроля за тем, какая программа и когда будет выполняться. В среде реального времени это недопустимо. Диспетчеризация, в основу которой положен принцип присвоения приоритета, и наличие ядра с приоритетом на прерывание позволяют разработчику прикладной программы полностью контролировать систему. Если наступает событие с высшим приоритетом, система прекращает обработку задачи с низшим приоритетом и отвечает на вновь поступивший запрос.

Сочетание описанных выше свойств создает мощную и эффективную среду исполнения в реальном времени.

Система разработки - набор средств, обеспечивающих создание и отладку приложения реального времени. Системы разработки (компиляторы, отладчики и всевозможные tools) работают, как правило, в популярных и распространенных ОС, таких, как UNIX и Windows.

Кроме того, многие операционные системы реального времени имеют и так называемые ***резидентные средства разработки***, исполняющиеся в среде самой операционной системы реального времени - особенно это относится к операционным системам реального времени класса "ядра".

Заметим, что функционально средства разработки операционных систем реального времени отличаются от привычных систем разработки, таких, например, как Developers Studio, TaskBuilder, так как часто они содержат:

- средства удаленной отладки,
- средства профилирования (измерение времен выполнения отдельных участков кода),
- средства эмуляции целевого процессора,
- специальные средства отладки взаимодействующих задач,
- а иногда и средства моделирования.

2 Ядро

Кроме свойств среды исполнения, необходимо рассмотреть также сервис, предоставляемый ядром ОС реального времени. Основой любой среды исполнения в реальном времени является ядро или диспетчер. Ядро управляет аппаратными средствами целевого компьютера:

- центральным процессором,
- памятью и устройствами ввода/вывода;
- контролирует работу всех других систем и программных средств прикладного характера.

Четкой границы между ядром (KERNEL) и операционной системой нет. Различают их, как правило, по набору функциональных возможностей.

Ядра предоставляют пользователю такие базовые функции, как

- планирование синхронизация задач,
- межзадачная коммуникация,
- управление памятью и т. д.

Операционные системы в дополнение к этому имеют

- файловую систему,
- сетевую поддержку,
- интерфейс с оператором,
- другие средства высокого уровня.

В системе реального времени диспетчер занимает место между аппаратными средствами целевого компьютера и прикладным программным обеспечением. Он обеспечивает специальный сервис, необходимый для работы приложений реального времени. Предоставляемый ядром сервис дает прикладным программам доступ к таким ресурсам системы, как, например, память или устройства ввода/вывода.

Ядро может обеспечивать сервис пяти типов:

Синхронизация ресурсов.

Метод синхронизации требует ограничить доступ к общим ресурсам (данным и внешним устройствам). Наиболее распространенный тип примитивной синхронизации - двоичный семафор, обеспечивающий избирательный доступ к общим ресурсам. Так, процесс, требующий защищенного семафором ресурса, вынужден ожидать до тех пор, пока семафор не станет доступным, что свидетельствует об освобождении ожидаемого ресурса, и, захватив ресурс, установить семафор. В свою очередь, другие процессы также будут

ожидать доступа к ресурсу вплоть до того момента, когда семафор возвратит соответствующий ресурс системе распределения ресурсов. Системы, обладающие большей ошибкоустойчивостью, могут иметь счетный семафор. Этот вид семафора разрешает одновременный доступ к ресурсу лишь определенному количеству процессов.

Межзадачный обмен.

Часто необходимо обеспечить передачу данных между программами внутри одной и той же системы. Кроме того, во многих приложениях возникает необходимость взаимодействия с другими системами через сеть. Внутренняя связь может быть осуществлена через систему передачи сообщений. Внешнюю связь можно организовать либо через датаграмму (наилучший способ доставки), либо по линиям связи (гарантированная доставка). Выбор того или иного способа зависит от протокола связи.

Разделение данных.

В прикладных программах, работающих в реальном времени, наиболее длительным является сбор данных. Данные часто необходимы для работы других программ или нужны системе для выполнения каких-либо своих функций. Во многих системах предусмотрен доступ к общим разделам памяти. Широко распространена организация очереди данных. Применяется много типов очередей, каждый из которых обладает собственными достоинствами.

Обработка запросов внешних устройств.

Каждая прикладная программа в реальном времени связана с внешним устройством определенного типа. Ядро должно обеспечивать службы ввода/вывода, позволяющие прикладным программам осуществлять чтение с этих устройств и запись на них. Для приложений реального времени обычным является наличие специфического для данного приложения внешнего устройства. Ядро должно предоставлять сервис, облегчающий работу с драйверами устройств. Например, давать возможность записи на языках высокого уровня - таких, как Си или Паскаль.

Обработка особых ситуаций.

Особая ситуация представляет собой событие, возникающее во время выполнения программы. Она может быть синхронной, если ее возникновение предсказуемо, как, например, деление на нуль. А может быть и асинхронной, если возникает непредсказуемо, как, например, падение напряжения. Предоставление возможности обрабатывать события такого типа позволяет прикладным программам реального времени быстро и предсказуемо отвечать на внутренние и внешние события. Существуют два метода обработки особых ситуаций - использование значений состояния для обнаружения ошибочных условий и использование обработчика особых ситуаций для прерывания ошибочных условий и их корректировки.

3 Методы планирования ОС

Важной частью любой ОСРВ является планировщик задач, чья функция - определить, какая из задач должна выполняться в системе в каждый конкретный момент времени.

К основным ***методам планирования*** обычно относят:

- циклический алгоритм (в стиле round robin),
- разделение времени с равнодоступностью (time sharing with fairness),
- кооперативная многозадачность.

Наиболее часто используемый в ОСРВ принцип планирования - приоритетная многозадачность с вытеснением. Основная идея состоит в том, что высокоприоритетная

задача, как только для нее появляется работа, немедленно прерывает (вытесняет) низкоприоритетную.

Однако диапазон систем реального времени весьма широк, начиная от полностью статических систем, где все задачи и их приоритеты заранее определены, до динамических систем, где набор выполняемых задач, их приоритеты и даже алгоритмы планирования могут меняться в процессе функционирования. Существуют, например, системы, где каждая отдельная задача может участвовать в любом из трех алгоритмов планирования или их комбинации (вытеснение, разделение времени, кооперативность). Кроме того, приоритеты тоже можно назначать по-разному. В общем случае алгоритмы планирования должны соответствовать критериям оптимальности функционирования системы. Однако, если для систем ***жесткого реального времени*** такой критерий очевиден "**ВСЕГДА и ВСЕ** делать вовремя", то для систем ***мягкого реального времени*** это может быть, например, минимальное максимальное запаздывание или средневзвешенная своевременность завершения операций. В зависимости от критериев оптимальности могут применяться алгоритмы планирования задач, отличные от рассмотренных. Например, может оказаться, что планировщик должен анализировать момент выдачи критичных по времени управляющих воздействий и запускать на выполнение ту задачу, которая отвечает за ближайшее из них (алгоритм earliest deadline first, EDF).

Хотя каждая задача в системе, как правило, выполняет какую-либо отдельную функцию, часто возникает необходимость в согласовании (синхронизации) действий, выполняемых различными задачами. Такая синхронизация необходима, в основном в следующих случаях:

1. Функции, выполняемые различными задачами, связаны друг с другом. Например, если одна задача подготавливает исходные данные для другой, то последняя не выполняется до тех пор, пока не получит от первой задачи соответствующего сообщения. Одна из вариаций в этом случае - это когда задача при определенных условиях порождает одну или несколько новых задач.
2. Необходимо упорядочить доступ нескольких задач к разделяемому ресурсу.
3. Необходима синхронизация задачи с внешними событиями. Как правило, для этого используется механизм прерываний.
4. Необходима синхронизация задачи по времени. Диапазон различных вариантов в этом случае достаточно широк, от привязки момента выдачи какого-либо воздействия к точному астрономическому времени до простой задержки выполнения задачи на определенный интервал времени. Для решения этих вопросов в конечном счете используются специальные аппаратные средства, называемые таймером.

Рассмотрим все четыре случая более подробно.

Связные задачи.

Взаимное согласование задач с помощью сообщений является одним из важнейших принципов операционных систем реального времени.

Здесь можно встретить такие понятия, как ***сообщение*** (message), ***почтовый ящик*** (mail box), ***сигнал*** (signal), ***событие*** (event), ***прокси*** (proxy) и т.п. Один и тот же термин для разных ОСРВ может обозначать разные вещи.

В дальнейшем будем называть сообщениями любой механизм явной передачи информации от одной задачи к другой (такие объекты, как семафоры, можно отнести к механизму неявной передачи сообщений). Объем информации, передаваемой в сообщении, может меняться от одного бита до всей свободной емкости памяти системы. Во многих ОСРВ компоненты операционной системы, также как и пользовательские задачи, способны принимать и передавать сообщения.

Сообщения могут быть ***асинхронными*** и ***синхронными***.

В первом случае доставка сообщений задаче производится после того, как она в плановом порядке получит управление.

А во втором случае циркуляция сообщений оказывает непосредственное влияние на планирование задач.

Например, задача, пославшая какое-либо сообщение, немедленно блокируется, если для продолжения работы ей необходимо дождаться ответа, или если низкоприоритетная задача шлет высокоприоритетной задаче сообщение, которого последняя ожидает, то высокоприоритетная задача, если конечно, используется приоритетная многозадачность с вытеснением, немедленно получит управление. Иногда сообщения передаются через буфер определенного размера (почтовый ящик). При этом, как правило, новое сообщение затирает старое, даже если последнее не было обработано.

Однако наиболее часто используется принцип, когда каждая задача имеет свою очередь сообщений, в конец которой ставится всякое вновь полученное сообщение. Стандартный принцип обработки очереди сообщений по принципу "первым вошел, первым вышел" (FIFO) не всегда оптимально соответствует поставленной задаче. В некоторых ОСРВ предусматривается такая возможность, когда сообщение от высокоприоритетной задачи обрабатывается в первую очередь (в этом случае говорят, что сообщение наследует приоритет пославшей его задачи). Сообщение может содержать как сами данные, предназначенные для передачи, так и указатель на такие данные. В последнем случае обмен может производиться с помощью разделяемых областей памяти, разделяемых файлов и т.п.

Общие ресурсы.

Ресурс - это общий термин, описывающий физическое устройство или область памяти, которые могут одновременно использоваться только одной задачей. Представим, например, что несколько задач пытаются одновременно выводить данные на принтер. На распечатке результата мы увидим страшную мешанину. Иногда может возникнуть ситуация, когда одна задача не вовремя прерывает другую, от которой зависит правильность выполнения исходной задачи (например, вытесняемая задача собирает информацию, которую использует вытесняющая задача). В результате может возникнуть серьезная ошибка. Упомянутые проблемы обусловлены ***времязависимыми ошибками*** (time dependent error), или ***гонками*** и характерны для многозадачных ОС, применяющих алгоритмы планирования с вытеснением (кстати, системы с разделением времени также относятся к категории вытесняющих). Приведенный пример показывает, что ошибки, обусловленные гонками

а) характерны для работы с любыми ресурсами, доступ к которым имеют несколько задач,

б) происходят только в результате совпадения определенных условий, а потому с трудом обнаруживаются на этапе отладки.

Критическими секциями называются участки кода программ, где происходит обращение к разделяемым ресурсам. Так как процессы обычно не имеют доступа к данным друг друга, а ресурсы физических устройств, как правило, управляются специальными задачами-серверами (драйверами), наиболее типична ситуация, когда гонки за доступ к глобальным переменным устраивают различные потоки, исполняемые в рамках одного программного модуля. Для того чтобы гарантировать, что критическая секция кода исполняется в каждый момент времени только одним потоком, используют механизм взаимоисключающего доступа, или попросту мутексов (Mutual Exclusion Locks, Mutex). Практически мутекс представляет собой разновидность семафора, который сигнализирует другим потокам, что критическая секция кода кем-то уже выполняется.

Если мутекс захвачен, то поток, пытающийся войти в критическую секцию, блокируется. После того, как мутекс освобождается, один из стоящих в очереди потоков

(если таковые накопились) разблокируется и получает возможность доступа к глобальному ресурсу.

В борьбе за ресурсы могут возникнуть следующие неприятности:

1. ***Смертельный захват*** (Deadlock). Обычно побочные эффекты этой ситуации называются более прозаично - <<зацикливание>> или <<зависание>>. А причина этого может быть достаточно проста - <<задачи не поделили ресурсы>>. Пусть, например, Задача А захватила ресурс клавиатуры и ждет, когда освободится ресурс дисплея, а в это время Задача В, успев захватить ресурс дисплея, ждет теперь, когда освободится клавиатура. В таких случаях рекомендуется придерживаться тактики <<или все, или ничего>>. Другими словами, если задача не смогла получить все необходимые для дальнейшей работы ресурсы, она должна освободить все, что уже захвачено, и повторить попытку только через определенное время. Другим решением, которое уже упоминалось, является использование серверов ресурсов.

2. ***Инверсия приоритетов*** (Priority inversion). Как уже отмечалось, алгоритмы планирования задач (управление доступом к процессорному времени) должны находиться в соответствии с методами управления доступом к другим ресурсам, а все вместе - соответствовать критериям оптимального функционирования системы. ***Эффект инверсии приоритетов*** является следствием нарушения гармонии в этой области. Ситуация здесь похожа на смертельный захват. Представим, что у нас есть
   - высокоприоритетная Задача А,
   - среднеприоритетная Задача В,
   - низкоприоритетная Задача С.

Пусть в начальный момент времени Задачи А и В блокированы в ожидании какого-либо внешнего события.

Допустим, получившая в результате этого управление Задача С захватила Семафор А, но не успела его отдать, как была прервана Задачей А.

В свою очередь, Задача А при попытке захватить Семафор А будет блокирована, так как этот семафор уже захвачен Задачей С.

Если к этому времени Задача В находится в состоянии готовности, то управление после этого получит именно она, как имеющая более высокий чем у Задачи С, приоритет.

Теперь Задача В может занимать процессорное время, пока ей не надоест, а мы получаем ситуацию, когда высокоприоритетная Задача А не может функционировать из-за того, что необходимый ей ресурс занят низкоприоритетной Задачей С.

Синхронизация с внешними событиями.

Известно, что применение аппарата прерываний является более эффективным методом взаимодействия с внешним миром, чем метод опроса. Разработчики систем реального времени стараются использовать этот факт в полной мере. При этом можно проследить следующие тенденции:

1. Попытка обеспечить максимально быструю и детерминированную реакцию системы на внешнее событие.
2. Стремление добиться минимально возможных периодов времени, когда в системе запрещены прерывания.
3. Подпрограммы обработки прерываний выполняют минимальный объем функций за максимально короткое время. Это обусловлено несколькими причинами:
   - Во-первых, не все ОСРВ обеспечивают возможность <<вытеснения>> во время обработки подпрограмм прерывания.
   - Во-вторых, приоритеты аппаратных прерываний не всегда соответствуют приоритетам задач, с которыми они связаны.

- В-третьих, задержки с обработкой прерываний могут привести к потере данных.

Как правило, закончив элементарно необходимые действия, подпрограмма обработки прерываний генерирует в той или иной форме сообщение для задачи, с которой это прерывание связано, и немедленно возвращает управление. Если это сообщение привело задачу в разряд готовых к исполнению, планировщик в зависимости от используемого алгоритма и приоритета задачи принимает решение о том, необходимо или нет немедленно передать управление получившей сообщение задаче. Разумеется, это всего лишь один из возможных сценариев, так как каждая ОСРВ имеет свои особенности при обработке прерываний. Кроме того, свою специфику может накладывать используемая аппаратная платформа.

Синхронизация по времени.

Как правило, в ОСРВ задается эталонный интервал (квант) времени, который иногда называют тиком (Tick) и который используется в качестве базовой единицы измерения времени. Размерность этой единицы для разных ОСРВ может быть разной, как, впрочем, разными могут быть набор функций и механизмы взаимодействия с таймером. Функции по работе с таймером используют

- для приостановки выполнения задачи на какое-то время,
- для запуска задачи в определенное время,
- для относительной синхронизации нескольких задач по времени и т.п.

Множество задач одновременно могут запросить сервис таймера, поэтому если для каждого такого запроса используется элемент в таблице временных интервалов, то накладные расходы системы по обработке прерываний от аппаратного таймера растут пропорционально размерности этой таблицы и могут стать недопустимыми. Для решения этой проблемы можно вместо таблицы использовать связный список и алгоритм так называемого дифференциального таймера, когда во время каждого тика уменьшается только один счетчик интервала времени.

Упражнения к лекции 2.

1. Что такое система исполнения реального времени? (дайте определение)
2. Какие требования предъявляются к системе исполнения реального времени?
3. Что такое система разработки реального времени? (дайте определение)
4. Из чего состоит система разработки реального времени?
5. Какие сервисы обеспечивает ядро ОСРВ?
6. Как осуществляется синхронизация ресурсов?
7. Как осуществляется межзадачный обмен?
8. Что значит разделение данных?
9. Как происходит обработка запросов внешних устройств?
10. Для чего нужна обработка особых ситуаций?

Литература к лекции 2.

2.1 Мартин ДЖ. Программирование для вычислительных систем реального времени. Пер. с англ. Изд-во "Наука", 1975.- 360с.

2.2 Тенанбаум Э. Современные операционные системы. пер. с англ. 2-е изд. – М.: СПБ.: Нижний Новгород: Питер, 2005. – 1037с.

2.3 Олифер В.Г. Олифер Н.А. Сетевые операционные системы. М.: СПБ.: Нижний Новгород: Питер, 2006. – 538с.

2.4 Грибанов В.П., Дробин С.В., Медведев В.Д. Операционные системы. - М.: Финансы и статистика, 1990. - 239 с.

2.5 Дейтел Х.М., Чофнес Р.Д. Операционные системы. пер. с англ. – М.: БИНОМ, 2006. – 704с.