Сергей Зыль

Проектирование, разработка и анализ программного обеспечения СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.06
ББК 32.973.26-018.2
З-96

Зыль С. Н.

З-96 Проектирование, разработка и анализ программного обеспечения систем реального времени. — СПб.: БХВ-Петербург, 2010. — 336 с.: ил. + CD-ROM

ISBN 978-5-9775-0559-8

Рассмотрены фундаментальные теоретические концепции систем реального времени, функциональная и информационная безопасность, разработка с использованием языка UML, а также верификация программного обеспечения. Описаны популярные программные платформы систем реального времени. Показаны возможности и особенности применения типовых инструментальных средств на примере конкретных прикладных задач. Компакт-диск содержит trial-версии программ, которые используются для части рассматриваемых практических упражнений (QNX SDP, aicas JamaicaVM, а также MinGW).

Для инженеров, технических руководителей, студентов и преподавателей вузов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 01.03.10.
Формат 60×90 $^{1}/_{16}$. Печать офсетная. Усл. печ. л. 21.
Тираж 1000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0559-8

Оглавление

Благодарности

По давней и очень доброй традиции, которая сложилась в технической литературе по вычислительной техники, хочу сказать слова благодарности в адрес людей, которые своими советами, комментариями, различной информацией внесли вклад в появление этой книги.

Примечание

Хотя я и не отличаюсь сентиментальностью, но, хотите верьте, хотите — нет, всегда внимательно и с удовольствием читаю благодарности авторов в книгах, которые изучаю.

Конечно, в первую очередь большую помощь мне оказывали мои коллеги Андрей Сеньков, Олег Большаков, Михаил Колесов, Владимир Махилёв, Павел Козлов, Андрей Докучаев — специалисты, обладающие экспертными знаниями в области операционных систем, языков программирования, аппаратных платформ и составляющие ядро и "золотой фонд" отдела разработки компании "СВД Встраиваемые Системы" (**www.kpda.ru**). К счастью для меня и к несчастью для них, они находятся в буквальном смысле слова "в шаговой доступности", и у меня есть бесценная возможность не только "пытать" их вопросами по широкому спектру проблем, что само собой, но и просить провести очередной "натурный эксперимент" для проверки или уточнения какой-либо мысли. Благо у них всегда есть пара-тройка стендов разработки, порой с весьма экзотической аппаратурой и конфигурацией.

Хотелось бы поблагодарить Дмитрия Рыжова — человека, знающего все или почти все о модельно-ориентированной разработке сложных систем. Несмотря на то, что он большую часть времени находится в командировках или скрывается в московском офисе

SWD Software (**www.swd.ru**), всевозможные способы удаленного взаимодействия, которые он осваивает так же быстро, как они появляются в природе, позволяют получать его исчерпывающие комментарии практически когда угодно.

Отдельное спасибо хотелось бы сказать Энди Вальтеру (Andy Walter) из германской компании aicas GmbH (**www.aicas.com**) и Мэтту Дэвису (Matt Davis) из английской компании IPL (**www.ipl.com**). Энди не только любезно предоставлял программное обеспечение для моих экспериментов и делился методическими материалами, но и терпеливо отвечал на все мои вопросы. Что касается Мэтта, то вполне достаточно сказать, что он — один из ведущих в мире специалистов по верификации программного обеспечения для ответственного применения. Особо хотелось бы отметить, что качество и оперативность технической поддержки компании aicas GmbH в моих глазах стали поистине эталоном технического сервиса, который должен обеспечиваться поставщиками программных продуктов.

Особую благодарность позвольте выразить Леониду Агафонову, управляющему директору компании SWD Software — именно его вера в необходимость данной книги убедила меня проделать эту работу. Ну и, конечно, спасибо Леониду за финансирование выхода этой книги в свет.

И, наконец, нужно упомянуть об Александре Варварике, генеральном директоре компании "СВД Встраиваемые Системы". Хотя обстоятельный разговор с ним на темы, не относящиеся к первостепенным задачам заказчиков и компании, практически исключен (во многом по причине его невообразимой способности перемещаться в пространстве), но зато другая его способность — на лету улавливать самую суть проблемы и в нескольких словах предлагать подход к ее решению — сделала его вклад в эту книгу весьма ценным.

Безусловно, этот список можно продолжать и продолжать. Это потому, что на мое сегодняшнее понимание целей, методов и средств создания программного обеспечения систем реального времени повлияли многие замечательные люди, с которыми мне посчастливилось работать и просто общаться. Спасибо вам, уважаемые коллеги!

Предисловие

Какой-то умный человек когда-то сказал, что всякий автор пишет ту книгу, которую хотел бы прочитать сам. Для книги, которую вы держите в руках, это утверждение верно на 100%. Она является результатом попытки систематизированного изложения знаний, почти ежедневно необходимых мне по работе, а также информации, без которой взгляд на проектирование, разработку и анализ программного обеспечения реального времени был бы лишен целостности и полноты.

Значительная часть книги построена на основе материалов моих выступлений на тематических конференциях, семинарах и тренингах. В итоге получилось представленное на ваш суд пособие.

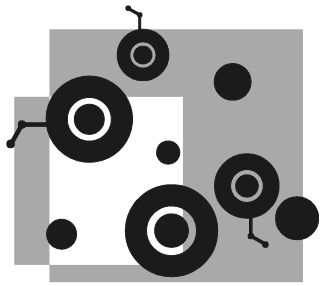
Признаюсь, что писать эту книгу было для меня удовольствием. Во-первых, это связано с тем, что ее материал касается вопросов, которые для меня представляют не только профессиональный интерес, но и в значительной степени являются хобби. Во-вторых, за семь лет, минувшие после выхода в свет моей первой книжки, моя аудитория существенно продвинулась вперед — тот материал, который был актуален раньше, т. е. механизмы и возможности операционных систем QNX, многие из прежних моих слушателей и читателей теперь знают гораздо лучше, чем я. Поэтому я с радостью смог позволить себе несколько уйти от технических деталей и показать читателю проблематику создания компьютерных систем реального времени, если так можно выразиться, с высоты "птичьего полета". Другими словами, в этой книге я всеми силами старался отвечать на вопрос "зачем?", а не на вопрос "как?". Что-то у меня получилось, а что-то — нет. Судить вам.

Материал книги построен так, чтобы быть максимально полезным для разработчиков программного обеспечения реального времени независимо от используемых ими продуктов и технологий. Конечно, многие вопросы рассматриваются на совершенно конкретных примерах, для которых я, разумеется, брал те технологии, с которыми больше всего знаком. Такой подход выбран не в целях рекламы той или иной программной продукции, а для того чтобы избежать оторванности от практической стороны обсуждаемых проблем.

Материал книги сведен в три части. *Первая часть* — чисто теоретическая — содержит фундаментальные концепции рассматриваемой предметной области. *Вторая часть* посвящена обсуждению предназначения и основных возможностей широко распространенных программных платформ реального времени — как инструментальных, так и времени исполнения. И наконец, *третья часть* представляет собой буквально пошаговые инструкции по установке конкретных инструментов с последующим решением небольших задач-примеров.

Надо отметить, что видимая "приземленность" третьей части нисколько не противоречит общему направлению книги. И вот почему. Для практических занятий третьей части выбраны продукты, которые, по моему субъективному мнению, весьма репрезентативны в соответствующих классах инструментов. И через пошаговое выполнение примеров я стремился показать не только то, какие задачи решаются с помощью этих инструментов, но и дать читателю возможность представить себе характер работ, выполняемых для решения этих задач. Ну и, придется признаться, еще хотелось спровоцировать вас, уважаемый читатель, сесть и попробовать самим "поиграть" с каким-либо из инструментов ☺.

На этом предлагаю закончить милую вводную беседу и, благословясь, приступить к нашим занятиям.

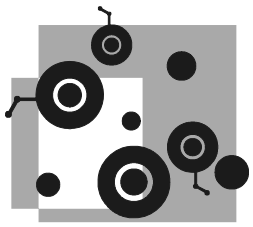


ЧАСТЬ I

Концепции

Первая часть нашего пособия содержит минимум теоретической информации, которая, на мой взгляд, необходима для целостного понимания такой предметной области, как разработка компьютерных систем реального времени. Мы рассмотрим компьютерные системы, которые называют системами реального времени, характеристики и механизмы реального времени, из каких этапов состоит разработка таких систем. Кроме того, мы рассмотрим подходы к управлению рисками, связанными с разработкой и эксплуатацией компьютерных систем реального времени, а также какие современные методологии могут повысить эффективность и продуктивность разработки.

ЗАНЯТИЕ 1



Механизмы реального времени

Целью этого занятия является объяснение базовых механизмов компьютерных систем реального времени и связанных с этими механизмами терминов.

В ходе занятия мы изучим:

- термины — система, реальное время и компьютерная система реального времени;
- какими базовыми функциональными характеристиками должны обладать операционные системы реального времени;
- какими параметрами обладают задачи реального времени и как классифицируют такие задачи;
- какие алгоритмы диспетчеризации могут использоваться для планирования задач реального времени;
- как используются вытисняемые системные вызовы для быстрой передачи управления обработчику прерывания или высокоприоритетной задаче;
- что такое инверсия приоритетов и как с ней бороться.

1.1. Что такое "компьютерная система реального времени"

Понимание термина *реальное время* применительно к вычислительной технике часто вызывает трудности. Почти любая статья

или презентация, посвященная компьютерным технологиям реального времени, начинается с объяснения этого термина. С вашего позволения я не буду оригинальным, и начну первое занятие с разъяснения своего понимания смысла слов, которые используются в названии этого курса.

В повседневной жизни словосочетание "реальное время" используют в качестве синонима термина "он-лайн". Например, от ведущих телепередач можно услышать "мы будем сообщать о ходе голосования в реальном времени". Поэтому нормальный человек, услышав слова "реальное время", представляет себе что-то очень быстрое. И поэтому в англоязычной технической литературе часто пишут кратко, но емко: *"real time* is not *real fast"*. Так что такое *реальное время* и что за "звери" эти *компьютерные системы реального времени*?

Для начала предлагаю вспомнить, определение термина *система*. "Википедия" [21] предлагает такое определение:

"*Систе́ма* (от греч. σύστημα, "составленный") — множество взаимосвязанных объектов, организованных некоторым образом в единое целое и противопоставляемое среде".

Кстати, термин *среда* так же очень важен для предмета нашего с вами обсуждения. Мне нравится определение "Википедии", поэтому предлагаю взять за основу именно его:

"*Среда* (в теории систем) — все объекты, не включенные в *систему*, с которыми *система* обменивается веществом, энергией и информацией".

Какая лаконичная, а главное — точная — формулировка!

Системы реального времени, в состав которых входит компьютер, как нетрудно догадаться, называют *компьютерными* или *вычислительными* системами реального времени. Пожалуйста, "зацепитесь" за то, что сказано в предыдущем предложении: когда говорят о *системах реального времени*, вовсе не подразумевают наличие в их составе какой-либо электронно-вычислительной техники! Компьютерные системы реального времени — только *частный случай* систем реального времени.

Теперь, думаю, можно процитировать так называемое каноническое определение систем (не обязательно компьютерных!) реаль-

ного времени, сформулированное математиком Дональдом Гиллесом (Donald Gillies):

"Система реального времени это такая система, в которой корректность вычислений зависит не только от логической корректности вычислений, но так же и от времени, за которое получен результат. Если временные ограничения системы не выдержаны, то считается, что возник сбой системы".

Примечание

В оригинале определение Гиллеса выглядит так: "A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred".

А теперь давайте рассмотрим представленную на рис. 1.1 схему некоторой абстрактной системы реального времени.

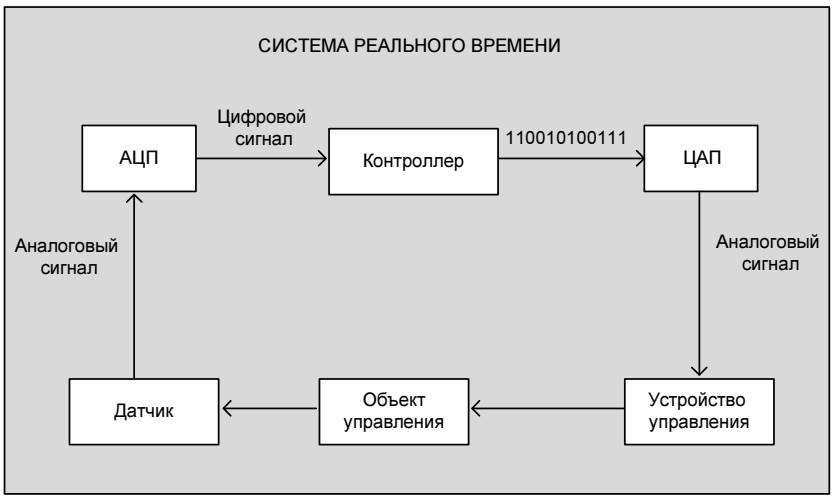


Рис. 1.1. Схема системы реального времени

На схеме изображены следующие элементы:

- *объект управления* — это то, что представляет интерес для потребителя, ради которого создавалась система. Объектом

управления может быть, например, газотурбинный двигатель, конвейер завода или что-либо еще;

- *контроллер* — это то, что "знает", что должен делать объект управления для эффективного и корректного выполнения своих функций в зависимости от каких-то факторов. Для управления сложным объектом в качестве контроллера, конечно же, используют компьютер. В этом случае для преобразования сигнала датчика в цифровую форму потребуется *аналогово-цифровой преобразователь* (АЦП), а для преобразования результата вычислений в управляющий электрический сигнал для устройства управления — *цифроаналоговый преобразователь* (ЦАП). Преобразователи могут быть периферийными устройствами контроллера или входить в состав средств измерения/управления;
- *датчик* — это то, с помощью чего контроллер "узнает" о том, что сейчас делает объект управления и каково состояние факторов, влияющих на выполнение объектом своих функций (в том числе, при необходимости, состояние среды). Обычно датчик помещают под воздействие измеряемого фактора, и он формирует электрический сигнал, соответствующий величине воздействия фактора. В качестве примеров можно привести датчики давления, температуры, задымленности, положения в пространстве и т. д.;
- *устройство управления* (УУ) — это то, с помощью чего контроллер воздействует на объект управления. Наиболее распространенным типом устройств управления являются различные приводы — шаговые двигатели, электромагнитные задвижки и т. п.

Представленная на рис. 1.1 система реального времени (в данном случае — *компьютерная*) работает так. Датчик считывает параметры объекта управления и передает на контроллер соответствующий электрический сигнал. С помощью АЦП сигнал преобразуется в цифровую форму и через некоторое *устройство ввода-вывода* (например, последовательный порт RS-232) поступает в компьютер. Программное обеспечение считывает полученную информацию и на ее основе, используя или не используя допол-

нительные сведения (таблицы данных, указания оператора и т. п.), выполняет необходимые расчеты и через устройство ввода-вывода передает команду на устройство управления (УУ). Цифровая команда с помощью ЦАП преобразуется в некоторый электрический сигнал, приводящий в действие устройство управления. И, наконец, устройство воздействует на объект управления.

Объект управления может быть весьма сложным и по структуре, и по выполняемым функциями. Датчиков также может быть весьма значительное количество, и не все они измеряют параметры объекта управления — они могут, разумеется, измерять *параметры среды* (определение термина см. ранее), в которой он работает, а также *параметры предметов*, на которые он воздействует. Например, контроллеру системы вентиляции и кондиционирования воздуха необходимо знать наружную температуру, а контроллеру станка могут понадобиться данные о результатах обработки детали станком.

Конечно, и *компьютерный модуль принятия решений* также может быть весьма сложным: представлять собой не один компьютер, а целый вычислительный кластер; включать в свой состав автоматизированные рабочие места (АРМ) операторов; хранить необходимые параметры и исходные данные в базах данных (БД); обеспечивать гарантированную степень надежности с помощью компьютеров "горячего" резерва и т. д. Кстати, компьютерные системы реального времени без человека-оператора называют системами автоматического управления (САУ), а с человеком-оператором — *автоматизированными системами управления* (АСУ).

Конечно, команда, выдаваемая на устройство управления, должна быть не только корректной, но и своевременной. Правильный, но полученный с опозданием результат вычислений может быть уже никому не нужным, например, опоздавшая команда контроллера антиблокирующей тормозной системы (ABS) автомобиля.

Для того чтобы окончательно поставить точку в отношении термина "реальное время", позволю себе привести еще одно определение. Его нельзя назвать формально точным, т. к. слово "система" используется в смысле "операционная система" (это в общем

случае не корректно). Но автора определения извиняет то, что его книга [11], в которой определение приводится, посвящена исключительно операционным системам. Это определение весьма четко показывает суть термина "реальное время". Вот это определение:

"Система реального времени — операционная система, способная отвечать на внешнее событие предсказуемым и правильным образом каждый раз, когда такое событие происходит".

Примечание

Пожалуйста, "споткнитесь" на слове "система" в этом определении! В данном случае оно означает не что иное, как "операционная система". Мы-то с вами просто ОБЯЗАНЫ чувствовать разницу между "системами" и "операционными системами"!

1.2. Особенности операционных систем реального времени

Компьютерный модуль принятия решения в системе реального времени как минимум должен уметь взаимодействовать с *внешними устройствами* (в первую очередь нас интересуют датчики и УУ) и обеспечивать формирование команды для УУ.

Рассмотрим взаимодействие с внешними устройствами. Как известно, программные инструкции выполняются процессором. При этом код инструкции, предписывающий процессору выполнить некоторую операцию, а также данные, над которыми выполняется операция, и результат операции размещаются в *оперативной памяти*. Все это хранится в сегментах памяти, называемых, соответственно, *сегментами кода* и *сегментами данных*. Таким образом, для того чтобы обработать данные из внешнего устройства, нужно каким-то образом поместить их в оперативную память и вызвать последовательность инструкций, необходимую для их обработки.

Внешние устройства могут быть подключены либо через порты ввода-вывода (например, UART или USB), либо к шине ввода-вывода. Во втором случае внешние устройства часто называют

периферийными. Схема, иллюстрирующая способы взаимодействия процессорного модуля с периферийными устройствами, представлена на рис. 1.2.

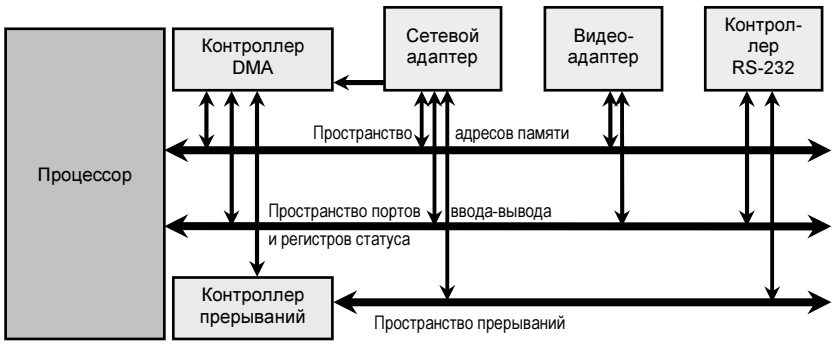


Рис. 1.2. Схема подключения периферийных устройств

В общем случае внешнее устройство может иметь:

- собственную память (например, видеопамять у видеоадаптера), которая может отображаться на некоторую область оперативной памяти компьютера так, что при поступлении данных в память устройства они автоматически оказываются в оперативной памяти. При работе с некоторыми устройствами для того, чтобы не использовать для копирования данных ресурсы процессора, может использоваться контроллер *DMA* (Direct Memory Access, прямой доступ к памяти);
- набор регистров, доступ к которым по чтению и записи осуществляется через механизм портов ввода-вывода.

Устройство может поместить данные, которые собирается передать в отображаемую память или порт ввода и сгенерировать электрический сигнал, уведомляющий о готовности данных к обработке. Контроллер прерываний возбуждает *прерывание* (по-английски *interrupt*) — электрический сигнал на определенных контактах процессора. В результате прерывания на процессоре прекращается выполнение текущей работы и вызывается *обработчик прерывания* — заранее определенный набор инструкций. Прерывание имеет номер, идентифицирующий устройство, — источник прерывания.

Нет нужды доказывать важность для системы реального времени того, чтобы обработчик прерывания начал работать как можно скорее. Увы, на практике это непросто. Ведь как минимум необходимо сохранить состояния регистров прерываемой программы, чтобы можно было впоследствии продолжить ее выполнение, и считать из ячеек оперативной памяти, именуемых *вектором прерывания*, адрес обработчика прерывания. Могут потребоваться и другие действия, например, переключение контекста между выполнявшейся программой и ядром операционной системы, считывающим адрес обработчика из вектора прерываний. Таким образом, между возникновением прерывания и началом выполнения обработчика в любом случае проходит какое-то время, в течение которого работает ядро операционной системы компьютера (рис. 1.3). Время между возбуждением прерывания и выполнением первой инструкции обработчика этого прерывания называют *задержкой прерывания* — это одна из ключевых характеристик реального времени для операционных систем.

Примечание

Иногда удобно использовать параметр сходный с задержкой прерывания, но другой — задержка реакции. *Задержка реакции* — время между моментом выставления устройством сигнала на шину и выполнением первой инструкции обработчика прерывания.

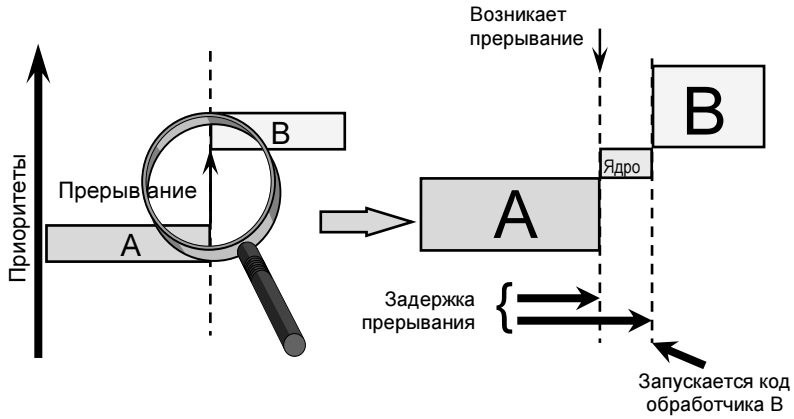


Рис. 1.3. Задержка прерывания

Вообще, операционные системы, как и любые другие плоды технического творчества человека, увы, не могут удовлетворить всем требованиям одновременно хотя бы потому, что требования часто бывают взаимоисключающими. Поэтому разработчики операционных систем, что бы ни говорили их продавцы, ориентируются на определенную область применения своих продуктов. В связи с этим операционные системы принято делить на две категории.

- *Операционные системы общего назначения* (иногда продавцы называют их операционными системами "мягкого реального времени"). Их основной целью является создание для пользователей комфортной среды работы и эффективное управление ресурсами компьютера, обеспечивающее минимизацию среднего времени выполнения задач.
- *Операционные системы реального времени* (иногда, для отличия от операционных систем "мягкого реального времени", их называют операционными системами "жесткого реального времени"). Их основной целью является обеспечение максимально быстрой реакции на прерывания и монопольное предоставление ресурсов процессора программе, выполняющей наиболее критичные для системы реального времени расчеты.

Отличия между операционными системами реального времени и операционными системами общего назначения в основном касаются организации обработки прерываний и подходов к диспетчеризации программ (об этом мы поговорим чуть позже). Есть и второстепенные отличия, которые, тем не менее, очень важны. Например, операционные системы общего назначения поставляются с достаточно большим количеством прикладных программ, необходимых для повседневного использования в офисе или дома (текстовые процессоры, редакторы изображений и т. д.), а инструменты разработки поставляются в виде отдельных продуктов (да и зачем они нужны подавляющему большинству пользователей?). Операционные системы реального времени, напротив, поставляются не то что с инструментами разработки, а обычно даже как составная часть таких инструментов — поставщик операционной системы реального времени часто даже во сне не сможет

предположить, что задумал сотворить покупатель. К тому же инструменты разработки для операционных систем реального времени обязательно должны включать хотя бы базовые средства анализа и верификации разработанного программного обеспечения — думаю, причины этого объяснять не нужно. Впрочем, нет, объясню, но позже, на третьем занятии.

Однако вернемся к задачам, стоящим перед компьютерным модулем принятия решений системы реального времени.

Программа может передавать в устройство управления данные и команды аналогично тому, как считывала информацию с датчиков, используя отображаемую память устройств и порты вывода.

В отношении взаимодействия с внешними устройствами остается добавить всего один штрих — часто считывание данных выполняют не по прерыванию, а через определенные промежутки времени, т. е. *по таймеру*. С точки зрения операционной системы принципиальной разницы выполнять обработку по прерыванию или по таймеру — нет: в любом случае выполняются некоторые, заранее определенные, действия. Кстати, поэтому и срабатывания таймера, и прерывания (и некоторые иные "происшествия") называют общим термином *событие* (по-английски *event*).

Примечание

На самом деле механизм таймеров непосредственно связан с аппаратными прерываниями. Точнее, с прерываниями от периодически срабатывающего аппаратного таймера компьютера. Временные интервалы, через которые операционная система умеет обрабатывать прерывания от аппаратного таймера, определяют точность службы времени операционной системы.

Очевидно, что зачастую компьютерный модуль принятия решения в системе реального времени может выполнять достаточно сложные операции, причем одновременно. К этим операциям, помимо формирования команд для различных устройств управления, можно отнести следующие:

- поиск необходимых для принятия решения данных в БД;
- отображение информации (возможно, мультимедийной) на дисплее человека-оператора АРМ;

- синхронизацию данных с компьютерами "горячего" резерва;
- обеспечение защиты информации от несанкционированного доступа;
- контроль целостности данных и программного обеспечения;
- другие функции.

Для того чтобы все нужные действия могли выполняться одновременно, необходимо, чтобы операционная система, управляющая ресурсами компьютера (процессором, оперативной и внешней памятью, периферийными устройствами), обеспечивала одновременное выполнение фрагментов программного кода, реализующего разные алгоритмы. Фрагменты кода, способные выполняться параллельно друг другу (а значит, и в достаточной степени независимо друг от друга), называют *задачами* (по-английски *task*). А операционные системы, поддерживающие возможность одновременного выполнения нескольких задач, называют *многозадачными операционными системами*.

Разумеется, если у компьютера всего один процессор и к тому же одноядерный, то многозадачность обеспечивается путем *разделения времени* — поочередного выполнения небольшой части каждой задачи. Это создает эффект "одновременности" выполнения задач, называемый *квазипараллелизмом*.

Реализация многозадачности в различных операционных системах может быть совершенно разная. Однако в последние 10—15 лет под влиянием стандарта POSIX сформировалось некое однообразие, предполагающее существование двух типов задач — *процессов* и *потоков*. Существующие в настоящее время операционные системы имеют разные модели процессов и потоков. Мне не хочется в этой книге обсуждать все возможные варианты и делать их сравнительный анализ — я буду говорить о классической модели, принятой в наиболее распространенных современных операционных системах:

- *Процесс* — это выполняющаяся программа. Процесс имеет уникальный в масштабах операционной системы *идентификатор* и владеет *ресурсами*. К ресурсам процесса относят его *образ* (загруженные в оперативную память сегменты кода и данных соответствующего исполняемого файла), созданные

им таймеры, открытые им файлы, атрибуты механизма защиты информации от несанкционированного доступа, переменные окружения и другую служебную информацию. Операционная система с помощью своего механизма *виртуальной памяти* и встроенного в процессор *модуля управления памятью* (Memory Management Unit — MMU) защищает образ процесса от "посягательства" других процессов. Обратите внимание на существенную *особенность процесса* — его образ размещается в защищенной от других процессов области оперативной памяти. Операционные системы, поддерживающие одновременную работу нескольких защищенных друг от друга процессов, часто называют *многопроцессными* или *с защитой памяти*. Некоторые операционные системы поддерживают *свопинг* — механизм выгрузки из оперативной памяти на жесткий диск неиспользуемых в текущий момент частей образов процессов, освобождая при этом место для образов активных процессов. Такой механизм позволяет запускать процессы с большими объемами кода и данных, но вносит задержки непредсказуемой продолжительности при необходимости срочно обратиться к откаченной на диск порции данных — это делает невозможным применение свопинга в операционных системах реального времени.

- *Поток* — это содержащийся в образе процесса последовательный набор программных инструкций, который может выполняться независимо от других таких же наборов инструкций. Поток имеет уникальный в пределах своего процесса идентификатор и параметры, к которым относится значение приоритета потока, дисциплина диспетчеризации, сигнальная маска и др. Часть образа процесса, которую занимают инструкции потока, называют *стеком*. По сути дела, процесс является контейнером потоков — реальные действия выполняют только потоки. Отсюда следствие — любой процесс содержит минимум один поток. Процесс, содержащий несколько потоков, называют *многопоточным*. Диспетчеризация потоков может осуществляться *в масштабе системы* (процессорное время делится поровну между потоками, имеющими равный приоритет, независимо от того, к каким процессам они принадлежат) или *в масштабе процесса* (процессорное время де-

лится между потоками каждого процесса в рамках времени, выделенного операционной системой процессу).

Примечание

Поток по-английски *thread*. Иногда слово thread переводят на русский язык дословно — нить. В англоязычной документации я встречал такое объяснение: "Нить (thread) — это отдельный поток (flow) управления".

На всякий случай напомню — реализация многозадачности в разных операционных системах может быть разной. Поэтому не поленитесь заглянуть в описание архитектуры той операционной системы, с которой вы работаете или собираетесь работать — найдете много интересного.

1.3. Классификация и характеристики задач реального времени

Таким образом, программная часть компьютерной системы реального времени представляет собой набор взаимодействующих между собой процессов или потоков. В теории реального времени для того, чтобы абстрагироваться от конкретных реализаций операционных систем используют универсальный термин *задача* (по-английски *task*). Мы тоже будем использовать этот термин в случаях, когда будем обсуждать общетеоретические вопросы. Итак, с точки зрения ограничений по времени выполнения задачи бывают следующих видов:

- *жесткого реального времени* — задачи, своевременность выполнения которых критична для всей системы;
- *мягкого реального времени* — задачи, исполнение которых желательно для системы, но некритично по времени;
- *прочие* — задачи, ко времени выполнения которых требований не предъявляется.

Примечание

У вас может возникнуть закономерный вопрос: "Кому нужны такие задачи, которые могут быть выполнены, а могут и не быть выпол-

нены?" Бывает, что очень даже нужны. Яркий представитель этого типа задач — поток idle ("морозильник") в ОСРВ QNX Neutrino. Это один из потоков менеджера процессов, который выполняется, когда и вправду больше нечему выполняться. Это необходимо для реализации функций энергосбережения.

Основным параметром задач реального времени является *предельное время выполнения* (по-английски *deadline*). Поскольку ни в какой операционной системе в силу разных причин нельзя добиться одинакового времени выполнения одних и тех же операций, на практике используют еще один важный критерий — *WCET* (от англ. Worst Case Execution Time — *худшее время исполнения*). Как нетрудно догадаться, для задач жесткого реального времени должно выполняться неравенство:

$$T_{WCET} \le T_{deadline} \tag{1.1}$$

Задачи мягкого реального времени хоть и не критичны для системы, но все-таки должны выполняться за разумное время. В качестве характеристики времени выполнения таких задач используют *ACET* (от англ. Average Case Execution Time — *среднее время выполнения*). Требования к задачам мягкого реального времени могут задаваться в виде диапазона времени или в виде вероятности выполнения за какое-то определенное время.

Кстати, WCET обратно пропорционально ACET. Таким образом, меры по минимизации WCET и меры по минимизации ACET для одной и той же задачи являются взаимоисключающими действиями. Поэтому забавно слышать рассуждения представителей тех или иных ведомств о единой операционной системе, которая позволяла бы и управлять аппаратурой, и обеспечивать канцелярское делопроизводство. Вот уж точно — в технике "серебряной пули" не существует.

В дальнейшем речь пойдет только о задачах жесткого реального времени.

Под *активизацией задачи* давайте договоримся понимать переход ее в состояние готовности к исполнению (состояние READY). Здесь, пожалуй, не лишним будет напомнить — задача может быть или в состоянии готовности к исполнению (READY), или в

заблокированном состоянии (BLOCKED). *Заблокированное состояние* — это состояние задачи при ожидании какого-то события. Процессорное время может предоставляться только готовым к исполнению задачам. Соответственно, в произвольный момент времени готовая к исполнению задача или выполняется на процессоре, или ждет своей очереди.

Итак, по характеру активизаций задачи можно разделить на три типа: периодические, апериодические и спорадические.

- *Периодические* — задачи, которые активизируются (т. е. становятся готовыми к исполнению на процессоре) через равные интервалы времени. Обычно это происходит, когда работа выполняется задачей циклически, т. е. задача ждет поступления порции данных, при поступлении которых выполняет их обработку и затем снова ожидает поступления уже следующей порции данных. Эти интервалы времени называют *периодами задачи*. В качестве примера можно привести периодическое считывание показаний датчика, выполняемое по таймеру. Период задачи *I* будем обозначать как T_i. Наверняка вы уже догадались, что в системе жесткого реального времени должно выполняться неравенство:

$$T_{WCET} \le T_{deadline} \le T_i \qquad (1.2)$$

- *Апериодические* — задачи, активизируемые со случайной периодичностью, ко времени выполнения которых требования не предъявляются (или это требования мягкого реального времени). Например, это могут быть запуски оператором АСУ профилактической диагностики, резервного копирования данных или выдача справочной информации.
- *Спорадические* — частный случай апериодических задач с известным минимальным интервалом между активизациями ($T_{\min}$) и жестким предельным временем исполнения. Например, запуск оператором АСУ программы, выполняющей безотлагательные действия для предотвращения возможной аварийной ситуации.

Можно заметить, что спорадическая задача имеет, в принципе, те же характеристики, что и периодическая — период и предельное

время выполнения. Только спорадическая задача может активизироваться реже (через любые интервалы времени, превышающие период) или не активизироваться вообще никогда, но надеяться на это разработчик программного обеспечения реального времени, конечно, не должен — он должен проектировать программное обеспечение исходя из того предположения, что спорадическая задача будет активизироваться по истечении каждого своего периода.

Обозначим фактический временной интервал активизации задачи I во время случая активизации J как T_{ij}. В таком случае для спорадической задачи должно выполняться следующее неравенство:

$$T_{ij} \geq T_{\min} \geq T_{deadline} \geq T_{WCET} \tag{1.3}$$

1.4. Алгоритмы диспетчеризации задач реального времени

Теперь, когда мы с вами разобрались во временны́х параметрах задач реального времени, можно перейти к обсуждению проблемы их *диспетчеризации* или *планирования*.

Диспетчеризацией задач называют порядок или алгоритм предоставления задачам ресурсов процессора. Диспетчеризация является одной из ключевых функций ядер операционных систем. Даже в микроядерных операционных системах, в которых максимально возможная часть традиционного для ядра функционала вынесена во внешние модули, диспетчеризация выполняется непосредственно ядром.

Основным критерием, на основании которого ядро операционной системы реального времени выбирает задачу для предоставления ресурсов процессора, является *приоритет*. Способность операционной системы при появлении готовой к выполнению задачи с приоритетом более высоким, чем приоритет текущей задачи, безболезненно прекратить выполнение текущей задачи и передать управление более приоритетной задаче называют *вытесняющей многозадачностью*.

На рис. 1.4 представлена UML-диаграмма времени, иллюстрирующая идею вытисняющей многозадачности (подробнее о языке визуального моделирования UML (Unified Modeling Language) см. *занятие 4*). С вашего позволения, прокомментирую диаграмму.

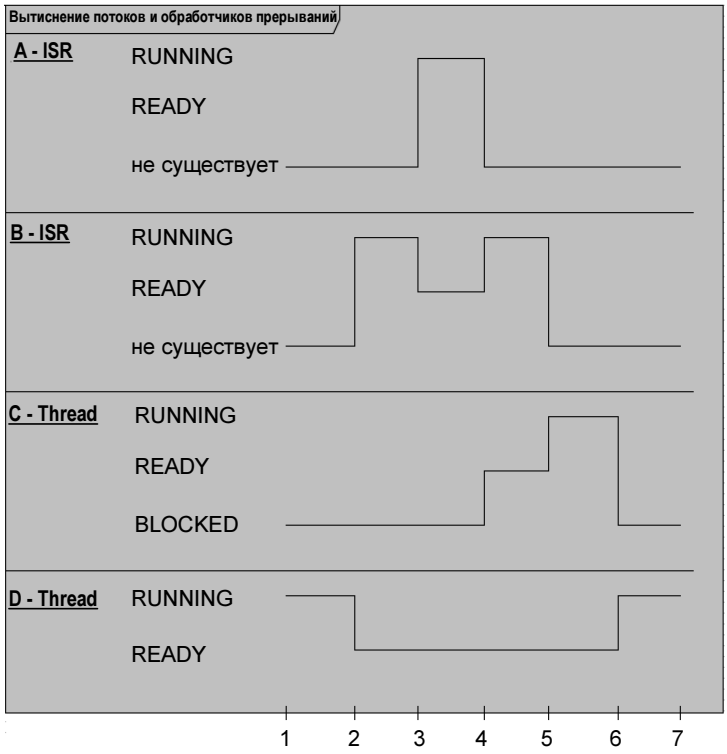


Рис. 1.4. Вытисняющая многозадачность

Примечание

Диаграмма, представленная на рис. 1.4, справедлива только в случае использования однопроцессорной системы, когда одновременное выполнение объектов диспетчеризации физически невозможно.

Мы видим четыре объекта диспетчеризации — обработчики прерываний (ISR — Interrupt Service Routine) A и B, а также потоки

C и D. Объекты диспетчеризации расположены вертикально согласно их приоритетам — чем выше объект, тем выше его приоритет. На момент времени 1 (шкала времени расположена внизу) существуют только потоки. При этом поток D выполняется на процессоре (RUNNING), а поток C заблокирован по ожиданию какого-то события или каких-то данных (BLOCKED).

В момент времени 2 возникает прерывание, скажем, от клавиатуры. В результате начнет выполнение обработчик B, при этом поток D будет вытеснен с процессора. Но при этом поток D будет в состоянии готовности к исполнению (READY).

Пока обработчик B делал свою работу (например, считывал сканкод нажатой клавиши), в момент времени 3 возникло более важное (например, от таймера) прерывание A. На процессор будет поставлен обработчик A, при этом обработчик B снимается с выполнения на процессоре (т. е. вытесняется) и будет ожидать освобождения процессора.

В момент времени 4 обработчик A завершил свою работу, подготовив при этом данные, которые ожидал поток C. Конечно, поток C стал готов к выполнению на процессоре, но процессор захватит обработчик B, поскольку у него приоритет выше, чем у потока C.

В момент времени 5 обработчик B закончил свою работу. Процессор захватит самый приоритетный поток — C, который будет выполняться до тех пор, пока в момент времени 6 ему не понадобится новая порция данных, и он снова не заблокируется по ожиданию этих данных. Это даст возможность потоку D продолжить выполнение.

Поддержка диспетчеризации на основе вытисняющей многозадачности является важной характеристикой реального времени для операционной системы. При этом следует обращать внимание, насколько "вложенным" (nested) может быть вытеснение потоков и, в особенности, обработчиков прерывания.

Пожалуйста, обратите внимание на огромную разницу между *приоритетом* и так называемым *nice-числом*! Это совершенно разные вещи. Приоритет — строгое указание ядру операционной системы на важность задачи. A nice-число, или *фактор уступчи-*

вости [10], как его иногда называют, — это вежливое пожелание, которое ядро должно бы учитывать при принятии решения о том, какой из задач передать управление. Наряду с nice-числом ядро принимает в учет такие параметры, как, например, время, в течение которого задача уже выполнялась, или количество используемой задачей памяти. Планировщики ряда операционных систем используют не приоритеты задач, а факторы уступчивости задач.

Как говорится, предупрежден — значит, вооружен ☺. Какой параметр использует диспетчер вашей операционной системы — приоритет или фактор уступчивости? Для ответа на этот вопрос необходимо заглянуть в описание архитектуры конкретной операционной системы.

Пожалуй, не лишним будет привести некоторую классификацию алгоритмов диспетчеризации, применяемых в операционных системах реального времени (рис. 1.5).

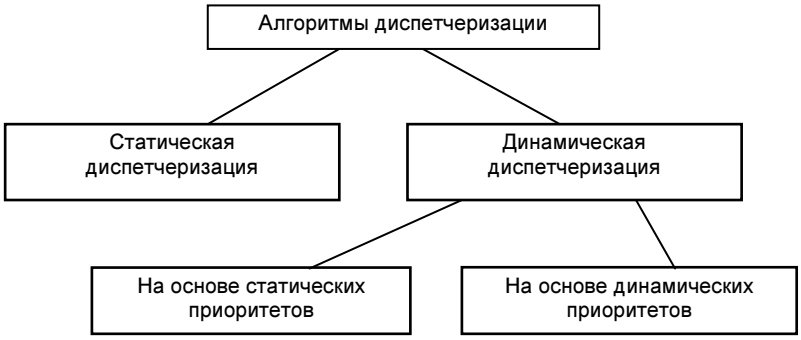


Рис. 1.5. Классификация алгоритмов диспетчеризации задач реального времени

Алгоритмы диспетчеризации делят на *статические* и *динамические*. При использовании статических алгоритмов порядок выполнения задач задается разработчиком на этапе проектирования и разработки системы. При использовании динамических алгоритмов решение о предоставлении процессора той или иной задаче принимается на этапе функционирования системы. Иногда в литературе статические алгоритмы называют *"офф-лайн"*-алго-