


Особенности программирования систем реального времени

Лекция 7

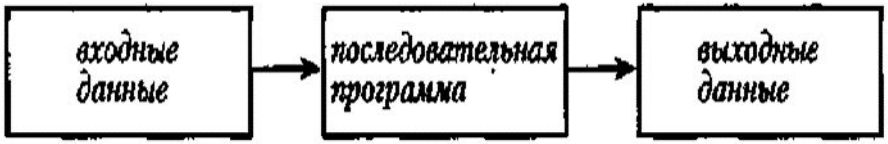
Методы программирования в реальном времени

1. Последовательное программирование и программирование задач реального времени
2. Среда программирования.
3. Структура программы реального времени.
4. Параллельное программирование, мультипрограммирование и многозадачность.

Последовательное программирование и программирование задач РВ

Программа представляет собой описание объектов - констант и переменных - и операций, совершаемых над ними. Таким образом, программа -это чистая информация. Ее можно записать на какой-либо носитель.

Программы можно создавать и анализировать на нескольких уровнях абстракции (детализации) с помощью соответствующих приемов формального описания переменных и операций, выполняемых на каждом уровне. На самом нижнем уровне используются непосредственное описание - для каждой переменной указывается ее размер и адрес в памяти. На более высоких уровнях переменные имеют абстрактные имена, а операции сгруппированы в функции или процедуры. Программист, работающий на высоком уровне абстракции, не должен думать о том, по каким реальным адресам памяти хранятся переменные, и о машинных командах, генерируемых компилятором



Последовательное программирование (sequential programming) - наиболее распространенный способ написания программ.

Понятие "последовательное" подразумевает, что операторы программы выполняются в известной последовательности один за другим. представляет собой описание объектов - констант и переменных - и операций, совершаемых над ними. Таким образом, программа -это чистая информация. Ее можно записать на какой-либо носитель. Целью последовательной программы является преобразование входных данных, заданных в определенной форме, в выходные данные, имеющие другую форму, в соответствии с некоторым алгоритмом - методом решения. Таким образом, последовательная программа работает как фильтр для исходных данных. Ее результат и характеристики полностью определяются входными данными и алгоритмом их обработки, при этом временные показатели играют, как правило, второстепенную роль. На результат не влияют ни инструментальные (язык программирования), ни аппаратные (быстродействие ЦП) средства: от первых зависят усилия и время, затраченные на разработку и характеристики исполняемого кода, а от вторых - скорость выполнения программы, но в любом случае выходные данные будут одинаковыми.

Программирование в реальном времени

Программирование в реальном времени (real-time programming) отличается от последовательного программирования - разработчик программы должен постоянно иметь в виду среду, в которой работает программа, будь то контроллер микроволновой печи или устройство управления манипулятором робота.

Программирование в реальном времени представляет собой раздел мультипрограммирования, который посвящен не только разработке взаимосвязанных параллельных процессов, но и временным характеристикам системы, взаимодействующей с внешним миром. Между программами реального времени и обычными последовательными программами, с четко определенными входом и выходом, имеются существенные различия:

1. Логика исполнения программы определяется внешними событиями.
2. Программа работает не только с данными, но и с сигналами, поступающими из внешнего мира, например, от датчиков.
3. Логика развития программы может явно зависеть от времени.
4. Жесткие временные ограничения. Невозможность вычислить результат за определенное время может оказаться такой же ошибкой, как и неверный результат ("правильный ответ, полученный поздно - это неверный ответ").
5. Результат выполнения программы зависит от общего состояния системы, и его нельзя предсказать заранее.
6. Программа, как правило, работает в многозадачном режиме. Соответственно, необходимы процедуры синхронизации и обмена данными между процессами.
7. Исполнение программы не заканчивается по исчерпании входных данных - она всегда ждет поступления новых данных.

Особенности программирования в реальном времени требуют специальной техники и методов, не использующихся при последовательном программировании, которые относятся к влиянию на исполнение программы внешней среды и временных параметров. Наиболее важными из них являются перехват прерываний, обработка исключительных (нештатных) ситуаций и непосредственное использование функций операционной системы (вызовы ядра из прикладной программы, минуя стандартные средства). Помимо этого при программировании в реальном времени используются методика мультипрограммирования и модель "клиент-сервер", поскольку отдельный процесс или поток обычно выполняют только некоторую самостоятельную часть всей задачи.

Среда программирования

Среда выполнения может варьироваться от мини-, персональных и одноплатных микрокомпьютеров и локальных шин, связанных с окружающей средой через аппаратные интерфейсы, до распределенных систем "клиент-сервер" с централизованными базами данных и доступом к системе высокопроизводительных графических рабочих станций.

В комплексной системе управления промышленными и технологическими процессами может одновременно использоваться все перечисленное оборудование.

Разнообразие аппаратной среды отражается и в программном обеспечении, которое включает в себя как программы, записанные в ПЗУ, так и комплексные операционные системы, обеспечивающие разработку и исполнение программ. В больших системах создание и исполнение программ осуществляются на одной и той же ЭВМ, а в некоторых случаях даже в одно время. Небольшие системы могут не иметь средств разработки, и программы для них должны создаваться на более мощных ЭВМ с последующей загрузкой в исполняющую систему. То же касается и микропрограмм, "зашитых" в ПЗУ оборудования производителем (firmware), - они разрабатываются на ЭВМ, отличной от той, на которой исполняются.

Первой задачей программиста является ознакомление с программной средой и доступными инструментальными средствами. Проблемы, с которыми приходится сталкиваться, начинаются, например, с типа представления данных в аппаратуре и программах, поскольку в одних системах применяется прямой, а в других - инверсный порядок хранения бит или байт в слове (младшие байты хранятся в старших адресах). Таких тонкостей очень много, и опытный программист знает, как отделить общую структуру данных и код от технических деталей реализации в конкретной аппаратной среде.

Важно как можно раньше выяснить функции, обеспечиваемые имеющейся средой, и возможные альтернативы. Например, микропроцессор Motorola 68000 имеет в своем наборе команд инструкцию test_and_set, и поэтому связь между задачами может осуществляться через общие области памяти. CVAX/VMS поддерживает почтовые ящики, и синхронизировать процессы можно с помощью механизма передачи сообщений.

В UNIX и других ОС связь между процессами наиболее удобно осуществлять через каналы. При разработке программ для среды UNIX следует стремиться, с одной стороны, максимально эффективно использовать ее

Принципы программирования в реальном времени

Из-за того, что многозадачные системы и системы реального времени разрабатываются коллективами программистов, необходимо с самого начала добиваться ясности, какие методы и приемы используются.

Структурирование аппаратных и программных ресурсов, то есть присвоение адресов на шине и приоритетов прерываний для интерфейсных устройств, имеет важное значение. Неправильный порядок распределения ресурсов может привести к тупиковым ситуациям. Определение аппаратных адресов и относительных приоритетов прерываний не зависит от разрабатываемой программы, поэтому должно выполняться на ранней стадии и фиксироваться в техническом задании. Если оно отложено до момента непосредственного кодирования, неизбежны конфликты между программными модулями и возникает риск тупиковых ситуаций.

Правильным практическим решением является использование в программе только логических имен для физического оборудования и его параметров и таблиц соответствия между ними и реальными физическими устройствами. При этом изменение адреса шины или приоритета устройства требует не модификации, а только новой компиляции программы. Разумно также использовать структурированное и организационно оформленное соглашение о наименовании системных ресурсов и программных переменных. То же относится и к наименованию и определению адресов удаленных устройств в распределенных системах.

Программы следует строить по принципам, применяемым в ОС, - на основе модульной и многоуровневой структуры, поскольку это существенно упрощает разработку сложных систем. Должна быть определена спецификация отдельных модулей, начиная с интерфейсов между аппаратными и программными компонентами системы. К основной информации об интерфейсах относится и структура сообщений, которыми будут обмениваться программные модули. Это не означает, что изменения в определении интерфейсов не могут вводиться после начала разработки программы. Но чем позже они вносятся, тем больше затрат потребует изменение кода, тестирование и т. д. С другой стороны, следует быть готовым к тому, что некоторые изменения спецификаций все равно будут происходить в процессе разработки программы, поскольку продвижение в работе позволяет лучше увидеть проблему.

Следует принимать во внимание эффективность реализации функций операционной системы. Нельзя считать, что способ, которым в операционной системе реализованы те или иные услуги, дан раз и навсегда. Для проверки того, насколько хорошо удовлетворяются временные ограничения, желательно провести оценку, например с помощью эталонных тестовых программ. Если результаты тестов неприемлемы, то одним из решений может быть разработка программ, замещающих

Структура программы реального времени

Разработка программы реального времени начинается с анализа и описания задачи. Функции системы делятся на простые части, с каждой из которых связывается программный модуль.

Например, задачи для управления движением робота-манипулятора можно организовать так: – считать с диска описание траекторий; – рассчитать следующее положение манипулятора (опорное значение); – считать с помощью датчиков текущее положение; – вычислить необходимый сигнал управления; – выполнить управляющее действие; – проверить, что опорное значение и текущее положение совпадают в пределах заданной точности; – получить данные от оператора; – остановить робота в случае нештатной ситуации (например, сигнал прерывания от аварийной кнопки).

Принципиальная особенность программ РВ - постоянная готовность и отсутствие условий нормального, а не аварийного завершения. Если программа не исполняется и не обрабатывает данные, она остается в режиме ожидания прерывания/события или истечения некоторого интервала времени. Программы РВ - это последовательный код, исполняющийся в бесконечном цикле. В каком-то месте программы есть оператор, приостанавливающий исполнение до наступления внешнего события или истечения интервала времени. Обычно программа структурируется таким образом, что оператор **end** никогда не достигается

while true do (*бесконечный цикл*) **begin** (*процедура обработки*) **wait event at #2,28** (*внешнее прерывание*) (*код обработки*) … **end**; (*процедура обработки*) **end**. (*выход из программы; никогда не достигается*)

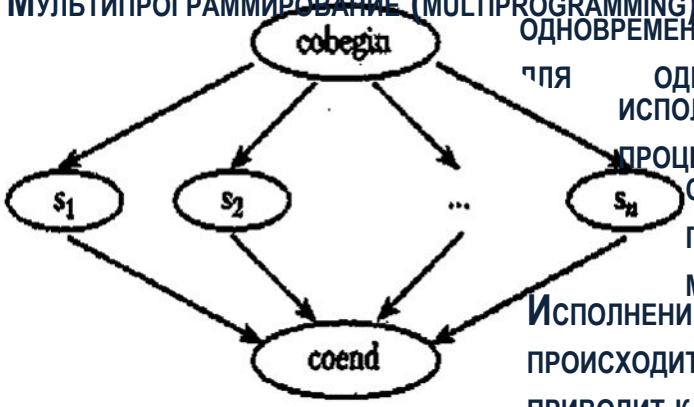
При разработке каждого программного модуля должны быть четко выделены области, в которых происходит обращение к защищенным ресурсам, - критические секции. Вход и выход из этих областей координируется каким-либо методом синхронизации или межпрограммных коммуникаций, например с помощью семафоров. В общем случае, если процесс находится в критической секции, можно считать, что данные, с которыми он работает, не изменяются каким-либо другим процессом. Прерывание исполнения процесса не должно оказывать влияния на защищенные ресурсы. Это снижает риск системных ошибок. Аналогичные предосторожности необходимо соблюдать и для потоков, порождаемых как дочерние процессы главного процесса. Разные потоки могут использовать общие переменные породившего их процесса, и поэтому программист должен решить, защищать эти переменные или нет. Для гарантии живучести программы нештатные ситуации, которые могут блокировать или аварийно завершить процесс, должны своевременно распознаваться и исправляться - если это возможно - в рамках самой программы.

В системах РВ различные процессы могут обращаться к общим подпрограммам, которые при простейшем решении связываются с соответствующими модулями после компиляции. При этом в памяти хранится несколько копий одной подпрограммы.

Оперативность исполнения является одним из наиболее важных параметров систем реального времени. Процессы должны

Параллельное программирование, мультипрограммирование и многозадачность

Программирование в РВ требует одновременного исполнения нескольких процессов или задач на одной ЭВМ. Эти процессы используют совместно ресурсы системы, но более или менее независимы друг от друга. Мультипрограммирование (multiprogramming) или многозадачность (multitasking) есть способ одновременного исполнения нескольких процессов. Такого эффекта можно добиться как для одного, так и для нескольких процессоров: процессы исполняются либо на одном, либо на нескольких связанных между собой процессорах. В действительности многие современные системы состоят из нескольких процессоров, связанных между собой либо сетью передачи данных, либо общей шиной. Для записи параллельных процессов можно использовать следующую нотацию




```
cobegin
x := 1;
 x := 2;
x := 3;
coend;
write (x);
```

Исполнение команд между ключевыми словами **cobegin** и **coend** происходит параллельно Пара операторных скобок **cobegin-coend** приводит к генерации потоков в рамках многозадачной системы.

Оператор **cobegin** не накладывает условий на относительный порядок исполнения отдельных процессов, а оператор **coend** достигается только тогда, когда все процессы внутри блока завершены. Если бы исполнение было последовательным, то окончательное значение переменной x было бы равно 3. Для параллельных процессов конечный результат однозначно предсказать нельзя; задачи выполняются, по крайней мере, с внешней точки зрения, в случайной последовательности. Поэтому окончательное значение x в приведенном примере может быть как 1, так и 2 или 3.

Иногда в технической литературе термин "параллельное программирование" используется как синоним мультипрограммирования. Однако эти понятия различаются по смыслу. Параллельное программирование - это абстрактный процесс разработки программ, который потенциально может исполняться параллельно, вне зависимости от программно-аппаратной среды. Иными словами, предполагается, что каждая задача реализуется на собственном виртуальном процессоре. С другой стороны, мультипрограммирование представляет собой практический способ исполнения нескольких программ на одном центральном процессоре или в распределенной вычислительной системе. Параллельное программирование более трудоемко, чем последовательное, поскольку способность человека следить за развитием связанных процессов, и исследовать их взаимодействие, ограничена. Программирование в РВ основано на параллельном программировании и включает в себя технику повышения эффективности и скорости исполнения программ - управление прерываниями, обработку исключений и непосредственное использование ресурсов ОС. Кроме того, программы РВ требуют специальных методов тестирования

2. Языки программирования реального времени

1. Требования к языкам программирования реального времени.
2. Языки разработки для систем реального времени.

Требования к языкам программирования реального времени

Программирование в реальном времени требует специальных средств, которые не всегда встречаются в обычных языках последовательного программирования. Язык или операционная система для программирования в реальном времени должны предоставлять следующие возможности: – описание параллельных процессов; – переключение процессов на основе динамических приоритетов, которые могут изменяться, в том числе и прикладными процессами; – синхронизацию процессов; – обмен данными между процессами; – функции, связанные с часами и таймером, абсолютное и относительное время ожидания; – прямой доступ к внешним аппаратным портам; – обработку прерываний; – обработку исключений.

Критерии при выборе языка для разработки приложения реального времени:

Некоторые компании разработали специальные языки для поддержки своих собственных аппаратных средств. Обычно они базируются на существующих языках - FORTRAN, BASIC - с расширениями, включающими функции реального времени, о чем свидетельствуют их названия типа "Process BASIC" и "Real-time FORTRAN". Некоторые языки не поддерживают программирования в реальном времени в строгом смысле, но они легко расширяются, например C и C++.

Языки разработки для систем реального времени

Ассемблер. Обеспечивает получение наивысшей производительности, прямой доступ к оборудованию, возможность вызова любых процедур на других языках. Однако, приложения получаются не переносимыми, объектно-ориентированный подход отсутствует. Обычно ассемблер используется только для написания небольших и четко локализованных фрагментов приложения, таких, как обработчики прерываний, драйверы устройств, критические по времени исполнения секции.

Язык программирования ADA. Первым полным языком программирования в реальном времени является ADA. Язык назван в честь Августы Ады Байрон, графини Лавлейс (Augusta Ada Byron, Countess of Lovelace, 1815-1852), которую можно считать первым программистом в истории - она писала программы для аналитической машины (механического компьютера, который никогда не был построен), спроектированной английским изобретателем Чарльзом Бэббиджем (Charles Babbage).

Язык ADA является полной средой разработки программ с текстовым редактором, отладочными средствами, системой управлениями библиотеками и т.д. Спецификации ADA закреплены американским стандартом ANSI/MIL-STD-1815A и включают средства контроля соответствия этому стандарту. Не допускаются диалекты языка - для сертификации компилятор должен правильно выполнить все эталонные тесты.

Структура языка ADA похожа на структуру языка Pascal, но его возможности значительно шире, в особенности применительно к СРВ. Процессу в ADA соответствует задача, которая выполняется независимо от других задач на выделенном виртуальном процессоре, то есть параллельно с другими задачами. Задачи могут быть связаны с отдельными прерываниями и исключениями, и работать как их обработчики.

Новым понятием, введенным в ADA, является пакет - модуль со своими собственными описаниями типов данных, переменных и подпрограмм, в котором явно указано, какие из программ и переменных доступны извне. Пакеты могут компилироваться отдельно с последующим объединением в один исполняемый модуль. Это средство поддерживает модульную разработку программ и создание прикладных библиотек. В начале 1990-х годов язык ADA был пополнен новыми функциями для объектно-ориентированного программирования и программирования в реальном времени.

Машинно-ориентированное программирование низкого уровня поддерживается ADA не достаточно эффективно - это следствие постулата, что все задачи должны решаться средствами высокого уровня. Например, для операций ввода/вывода в ADA используются прикладные пакеты с заранее определенными функциями для управления аппаратными интерфейсами и доступа к внешним данным.

Основным недостатком ADA является его сложность, которая делает язык трудным для изучения и применения. Существующие компиляторы являются дорогостоящими продуктами и требуют мощных процессоров. До сих пор ADA не

Языки C и C++

Язык программирования C стал популярным для всех приложений, требующих высокой эффективности, в частности для программ РВ. Для обычных микропроцессоров, используемых в системах управления, имеются С-компиляторы и системы разработки многих производителей. В промышленности существует тенденция к широкому применению языка C и ОС UNIX, которая сама написана на C, поскольку приложения, написанные на C, машинно-независимы и требуют не больших усилий для адаптации к работе в различной аппаратной среде. Философией C является разбиение программ на функции. C - слаботипизированный язык и позволяет программисту делать почти все вплоть до манипуляции с регистрами и битами. Такая свобода делает язык небезопасным, т.к. компилятор не может проверить, являются ли подозрительные операции умышленными или нет. Небольшое количество заранее определенных функций и типов данных делает программы легко переносимыми между разными системами. C поддерживает как структурированный, так и плохой стиль программирования, оставляя ответственность за качество разработки на программисте. Стиль программирования приобретает особое значения при сопровождении программ: плохо написанная и откомментированная программа на C - такая же загадка, как и ассемблерский код. Язык C регламентирован международным стандартом ISO 9899.

Язык C предпочтителен для написания программ с обращениями к функциям ОС, так как он обладает отличной совместимостью между логикой определения переменных и синтаксисом обращения к системе. Поскольку наиболее распространенные ОС в приложениях автоматического управления процессами основываются на UNIX, язык C является почти вынужденным выбором при разработке программ. C обеспечивает получение высокой производительности за счет хорошо разработанных оптимизирующих компиляторов, которые для современных процессоров часто дают код более эффективный, чем написанный на ассемблере. Язык C дает прямой доступ к оборудованию и возможность вызова процедур на других языках. Приложения получаются переносимыми (особенно, если ОСРВ поддерживают одинаковый стандарт, например POSIX), однако, объектно-ориентированный подход на уровне языковых конструкций отсутствует.

Язык C++ представляет собой значительно более мощный инструмент, чем C. В C++ значительно улучшена абстракция данных с помощью понятия класса, похожего на абстрактный тип данных с четким разделением между данными и операциями. Классы C++ значительно легче использовать на практике, поскольку C++ поддерживает объектно-ориентированное программирование и поэтапное уточнение типов данных. Главным преимуществом языка C++ является его способность поддерживать разработку легко используемых библиотек программ. Программирование в реальном времени непосредственно в C++ не поддерживается, но может быть реализовано с помощью специально разработанных программных модулей и библиотек классов.

C++ включает язык C как подмножество и наследует все его положительные качества. C++ добавляет поддержку

Языки программирования высокого уровня

Java. Как язык интерпретируемого типа, имеет очень низкую эффективность получаемого кода. Доступ к оборудованию и вызовы процедур на других языках - только посредством библиотечных функций (обычно написанных на C). Java обеспечивает наивысшую переносимость приложения на уровне двоичного кода и является объектно-ориентированным языком.

BASIC. является простейшим среди языков программирования высокого уровня. BASIC имеется почти на всех мини- и микрокомпьютерах. Программа на BASIC может компилироваться, но чаще она интерпретируется, то есть каждая команда транслируется в машинные коды только в момент ее выполнения. BASIC удобен для разработки небольших прикладных задач в составе крупных систем. BASIC является наилучшим средством для непрофессиональных программистов, которым требуется быстро решить частную задачу. Командные языки, основанные на BASIC, имеются во многих системах промышленной автоматики.

FORTRAN - первый язык программирования высокого уровня, который способствовал распространению и практическому применению ЭВМ. В целом FORTRAN имеет ограниченные возможности определения типа, весьма сложный способ работы с нечисловыми данными и не содержит многих важных функций языков реального времени. Новые версии FORTRAN заимствовали некоторые возможности из других языков и поддерживают более развитые структуры данных. Благодаря тому, что язык имеет устойчивое применение в научных приложениях, нередко данные в СРВ обрабатываются существующими FORTRAN-программами, а новые программы анализа и статистики пишутся на FORTRAN. При этом основной проблемой является координация передачи информации между БД РВ и прикладными модулями, написанными на FORTRAN. Такая координация обычно выполняется ОС. FORTRAN не рекомендуется для написания драйверов устройств или модулей на уровне ОС.

Pascal был разработан как дидактический язык для обучения хорошей технике программирования. Он в настоящее время используется во множестве разнообразных приложений. Успех Pascal, как в случае BASIC, основан на распространении микро- и персональных компьютеров, на которых он широко используется. **Язык Modula-2** был разработан специально для программирования встроенных промышленных и научных вычислительных СРВ. Modula-2 обладает большим количеством функций и синтаксических конструкций.

Языки четвертого поколения (CASE средства). Средства CASE (Computer Aided Software Engenering) получили широкое распространение при разработке приложений реального времени в силу большой сложности последних. Языки «четвертого поколения» представляют собой формализованный способ описания объектов, их свойств и взаимоотношений между собой. По этому формальному описанию «компилятор» строит текст приложения на языке более низкого уровня (обычно предоставляется выбор между C/C++/Java). Затем этот текст можно скомпилировать уже «обычным» компилятором.

Программирование асинхронной и синхронной обработки данных

1. Обработка прерываний и исключений.
2. Программирование операций ожидания.
3. Внутренние подпрограммы операционной системы.
4. Приоритеты процессов и производительность системы.
5. Тестирование и отладка.

Обработка прерываний и исключений

СРВ соединены с внешней средой через аппаратные интерфейсы. Доступ к интерфейсам и внешним данным осуществляется либо по опросу, либо по прерыванию.

При опросе программа должна циклически последовательно проверять все входные порты на наличие у них новых данных, которые затем считываются и обрабатываются. Очередность и частота опроса определяют время реакции системы реального времени на входные сигналы. Опрос является простым, но неэффективным методом из-за повторяющихся проверок входных портов.

При получении данных по прерыванию интерфейсное устройство, привлекает внимание центрального процессора, посылая ему сигнал прерывания через системную шину. По отношению к текущему процессу прерывания являются асинхронными событиями, требующими немедленной реакции. Получив сигнал прерывания, процессор приостанавливает исполнение текущего процесса, сохраняет в стеке его контекст, считывает из таблицы адрес программы обработки прерывания и передает ей управление. Эта программа называется обработчиком прерывания. Другой вариант: планировщик выбирает из очереди ожидания этого события или прерывания следующий процесс и переводит его в очередь готовых процессов.

Когда процессор передает управление обработчику прерываний, он обычно сохраняет только счетчик команд и указатель на стек текущего процесса. Обработчик прерываний должен сохранить во временных буферах или в стеке все регистры, которые он собирается использовать, и восстановить их в конце. Эта операция критична по времени и, как правило, требует запрета прерываний для того, чтобы избежать переключения процессов во время ее выполнения. При управлении прерываниями время реакции должно быть как можно меньше. Если система должна обслуживать много одновременных прерываний, вновь поступающие прерывания будут ждать в очереди, пока процессор не освободится.

Программа обработки прерывания должна быть предельно компактной (длина кода) и короткой (время выполнения) и выполнять лишь минимально необходимые операции, например, считать входные данные, сформировать сообщение и передать другой программе, извещая ее, что произошло прерывание и требуется дальнейшая обработка.

Реакция на исключения (exceptions) похожа на обработку прерываний. Исключениями называются нештатные ситуации, когда процессор не может правильно выполнить команду. Примером исключения является деление на ноль или обращение по несуществующему адресу. В литературе применяются термины trap, fault, abort (не путать с "взаимным исключением" - mutual exclusion).

Обычно ОС обрабатывает исключения, прекращая текущий процесс, и выводит сообщение. Приемлемая при последовательной интерактивной многопользовательской обработке, внезапная остановка процесса в системах реального времени должна быть абсолютно исключена. В СРВ все возможные исключения должны анализироваться заранее с

Программирование операций ожидания

Процесс реального времени может явным образом ждать истечения некоторого интервала (относительное время) или наступления заданного момента (абсолютное время). Соответствующие функции обычно имеют следующий формат: **wait (n) и wait until (время)** где **n** - интервал в секундах или миллисекундах, а переменная "время" имеет формат часы, минуты, секунды, миллисекунды.

Когда выполняется одна из этих функций, операционная система помещает процесс в очередь ожидания. После истечения/наступления заданного времени процесс переводится в очередь готовых процессов.

Распространенный, но не лучший метод организации временной задержки - цикл, контроль системного времени в цикле занятого ожидания

```
repeat (*холостой ход*)
until (time = 12:00:00);
```

Как правило, подобные активные циклы ожидания представляют собой бесполезную трату процессорного времени, и их следует избегать. Однако имеются исключения.

В системе, где аналого-цифровое преобразование занимает 20 мкс, а операция переключения процессов - 10 мкс, более экономно организовать ожидание на 20 мкс перед тем, как считать новые данные, чем начинать процедуру переключения процессов, неявно подразумеваемую "хорошей" операцией ожидания. Каждый случай требует индивидуального подхода - для этого обычно нужно хорошее знание системы и развитое чутье.

Важной особенностью процессов, запускаемых периодически, является накопленная ошибка времени. Это связано с тем, что процесс из очереди ожидания события опять попадает в очередь не уже готовых процессов и должен ждать некоторый случайный интервал времени прежде, чем получит управление. Требуемое и фактическое время пробуждения процесса не совпадают. Ошибки ожидания накапливаются, если это время рассчитывается так новое время пробуждения = время начала ожидания + интервал По такому алгоритму работает холостой цикл "ждать 10 секунд". Накопленная временная ошибка представляет собой сумму времени, проведенного в очереди, и времени, необходимого для непосредственного исполнения.

Правильное решение получается, если отсчет ведется от момента предыдущего пробуждения

новое время пробуждения = время предыдущего пробуждения + интервал

Таким образом, относительное время преобразуется в абсолютное. На практике необходимы две команды

```
wait until (ref_time);
```

Внутренние подпрограммы операционной системы

Типичная ситуация при программировании в РВ - непосредственное обращение к подпрограммам ОС из-за того, что в используемом языке программирования отсутствует эквивалентное средство.

Обращения к функциям ОС также необходимы при работе в сетевой и распределенной среде. Операционная система отвечает за все обслуживание прикладных задач, включая файловые и сетевые операции. Простое обращение к ОС может привести к сложной последовательности действий для доступа к удаленной базе данных, включая все сопутствующие проверки и операции управления, избавляющие прикладную программу от лишних деталей. Интерфейс ОС делает выполнение таких операций более прозрачным и упрощает написание сложных программ.

Многие языки программирования высокого уровня, например С, обеспечивают интерфейс с операционной системой для непосредственного вызова ее модулей из исполняемых процессов. Существуют различные виды программных интерфейсов с операционной системой непосредственные вызовы, примитивы и доступ через библиотечные модули.

Непосредственные (системные) вызовы осуществляются с помощью конструкции языка высокого уровня, которая передает управление подпрограмме, являющейся частью операционной системы. Необходимые параметры передаются списком, как при обычном обращении к подпрограмме. После завершения системной процедуры результат возвращается вызывающей программе.

Так как в многозадачной среде системные программы и примитивы могут вызываться одновременно разными процессами, их код всегда реентерабелен. Это позволяет избежать конфликтов при прерывании системной программы другим запросом, требующим ту же услугу из другого контекста.

В некоторых случаях для доступа к внутренним ресурсам операционной системы можно использовать библиотечные модули. Эти модули уже предварительно откомпилированы, и их остается только связать с основной программой. Необходимо проверить по документации системы требуемые параметры, а также механизмы их передачи и редактирования связей в языке высокого уровня.

Приоритеты процессов и производительность системы

Многозадачная ОС РВ должна допускать назначение приоритетов исполняемым процессам. Обычно приоритеты являются динамическими, что означает, что во время исполнения они могут изменяться как самими процессами, так и ОС. Обычно существуют определенные ограничения и механизмы контроля, которые определяют, кто и как может менять приоритеты. Назначение приоритетов оказывает серьезное влияние на работу системы в целом. Наиболее важные процессы или процессы, время реакции которых жестко ограничено, получают более высокий приоритет. К последним относятся обработчики прерываний. Задачи, выполняющие менее важные действия, например печать, получают более низкий приоритет. Необходимо обращать внимание на соглашения, используемые в системе относительно того, связан ли более высокий приоритет с большим или меньшим числом. Приоритеты имеют относительное значение и оказывают влияние только тогда, когда существуют процессы с разными приоритетами.

В СРВ реакция на прерывания отделена от вычислений, требующих значительных ресурсов процессора. Как только происходит событие или прерывание, его обработчик немедленно включается в очередь готовых процессов. Программы обработчиков прерываний обычно компактны, так как они должны обеспечивать быструю реакцию, например ввод новых данных и передавать управление более сложным процессам, интенсивно потребляющим ресурсы процессора, которые исполняются с более низким приоритетом.

Производительность СРВ значительно труднее поддается оценке, поскольку программа РВ зависит от поведения окружающей среды, то есть управляемых технических процессов. Общая производительность системы должна быть достаточной для того, чтобы выполнять все операции и выдавать результаты за установленное время. В развитых и сложных ОС, таких как UNIX, и в еще большей степени в распределенных ОС, доступ к большинству функций (ввод/вывод, сетевая поддержка и т.д.) происходит через системные вызовы или механизм удаленного вызова процедур. В прикладных программах для вызова системных функций используется довольно простая нотация, за которой, как правило, стоит длинная последовательность действий ОС. Если между двумя процессами, исполняющимися в разных узлах сети, организован программный канал, то считывание одного символа из этого канала требует целой серии операций в обоих узлах.

Поскольку на эти операции обычно наложены жесткие ограничения по времени, необходим глубокий предварительный анализ прежде, чем принимать проектное решение. Если локальная сеть используется не только задачами РВ, ями, то от количества и активности пользователей, зависит и ее общая нагрузка.

Многозадачные ОС имеют команды, показывающие в каждый момент все активные процессы, их текущий статус и долю в потреблении ресурсов процессора. Выявление процессов, занимающих слишком большую долю процессорного времени, может быть хорошей отправной точкой для поиска узких мест и оптимизации характеристик системы. Нет ничего плохого в том, если некоторые процессы загружают процессор больше, чем другие, но разработчик системы должен иметь

Тестирование и отладка

Доказательство правильности работы программы является обязательным шагом в ее разработке. Необходимо проверить, что программа выполняет свои функции без ошибок. Визуальные и формальные методы позволяют выявить только ограниченное количество ошибок.

На практике это означает, что формальная теория тестирования имеет мало смысла, а основную роль играет собственный опыт и "народные программистские" предания. Реальное тестирование проводится в "боевых" условиях.

Выявлять ошибки трудно - многие из них проявляются спорадически и их нельзя воспроизвести по желанию. Никакое доказательство не может гарантировать, что программа полностью свободна от ошибок, и никакие тесты не могут убедить, что выявлены все ошибки. Цель тестирования - найти как можно большее число ошибок и гарантировать, что программа работает с разумной надежностью. Один из создателей теории операционных систем, Эдсгер Дейкстра (Edsger Dijkstra), заметил: "Тестирование может доказать только наличие ошибок, но не их отсутствие".

Тщательный тест требует соответствующей разработки и подготовки; необходимо сочетание практических и аналитических тестов. Сначала тестовые процедуры и данные, ожидаемые результаты описываются в специальном документе. В процессе тестирования ведется журнал испытаний, который затем сравнивается со спецификацией тестов. Желательно, чтобы коллектив разработчиков системы отличался от того, который будет определять процедуры испытаний и проводить их.

При тестировании систем реального времени существует дополнительная сложность из-за большого количества возможных взаимосвязей между задачами. Вероятность внесения новой ошибки при исправлении старой очень велика - имеющийся опыт разработки программ размером свыше 10000 строк дает вероятность в пределах от 15 до 50%.

Существует два основных метода тестирования – исчерпывающий, и на примерах. При исчерпывающем тестировании проверяются все возможные комбинации входных и выходных данных. Очевидно, что этот метод можно использовать лишь в случае, если число таких сочетаний невелико.

Тестирование и отладка

Метод испытаний на примерах используется наиболее часто. Выбирают репрезентативное число сочетаний входа и выхода. Тестовые данные должны также включать крайние значения, например, находящиеся за пределами допустимого диапазона. Тестируемый модуль должен правильно распознать и обработать эти данные.

В многозадачных системах программные модули вначале тестируются отдельно. Во время такого тестирования должно быть проверено, что каждая строка программы выполняется хотя бы один раз. Иными словами, если программа содержит команды ветвления типа "if..then..else", то тестовые данные должны обеспечить выполнение обеих ветвей.

На этой фазе тестирования обычно полезны отладчики. Они позволяют непосредственно просматривать и изменять регистры процессора и области памяти при исполнении машинного кода. Отладчик вставляет в машинный код программы точки останова, в которых можно проверить состояние регистров и переменных и сравнить их со значениями, требуемыми логикой процесса. Однако с ростом сложности ОС и расширением функциональности системных вызовов, код которых обычно неизвестен программисту, использование отладчика может оказаться мало полезным, не позволяет полностью оценить взаимодействие между несколькими параллельными процессами. Однако отладчики являются полезными и необходимыми средствами при разработке программ на ассемблере.

Только после того как все модули были проверены по отдельности и все обнаруженные ошибки исправлены, можно приступать к параллельному исполнению для отладки взаимодействия. Многочисленные взаимосвязи программных модулей могут привести к ошибкам в системе, даже если отдельные модули работают правильно. Общая работа системы - время обработки прерываний, производительность при разной нагрузке - проверяется на основе тестовой спецификации. Особое внимание следует обратить на функции, обеспечивающие надежность и безопасность системы.

Если система включает в себя обработку прерываний и исключений, то необходимо проверить корректность соответствующей реакции. Имитация ошибочных ситуаций позволяет оценить их последствия для системы и ее поведение в этом случае.

Результаты тестов отдельных модулей и комплексной отладки заносятся в протокол испытаний, и на его основе вносятся необходимые исправления. Ошибки тем труднее исправляются, чем позже они были обнаружены. Расходы на тестирование - это инвестиции не только в качество системы, но и в ее общую экономическую эффективность,

Thank You!