Системы реального времени

Real-time systems

Тема 1 – Введение в СРВ

§ 1.1 Определения СРВ
§ 1.2 Типы СРВ
§ 1.3 Структура СРВ
§ 1.4 Закрепление материала
§ 2 - Введение в ОС РВ
§ 2.1 Требования к ОС РВ
§ 2.2 Различия между ОС РВ и ОС общего назначения
§ 2.3 Характеристики ОС РВ
§ 2.4 Механизмы РВ
§ 2.5 Архитектура ОС РВ
§ 2.6 Классы СРВ
§ 3 Планирование задач и процессов ОС РВ
§ 3.1 Строение ОС РВ
§ 3.2 Функции ядра ОС РВ
§ 3.3 Задачи, процессы, потоки

Тема 1 – Введение в СРВ.

Система реального времени – это система, которая способна обеспечить требуемый уровень сервиса в определённый промежуток времени.
Главный критерий эффективности – обеспечение временных характеристик.

§ 1.1 Определения СРВ

Система реального времени – это аппаратно-программный комплекс, реагирующий в предсказуемые времена на непредсказуемый поток внешних событий. Это означает, что:

А) Система должна успеть отреагировать на событие, произошедшее на объекте, в течение времени, критического для этого события. Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть предсказано (вычислено) при создании системы. Отсутствие реакции в предсказанное время считается ошибкой для систем реального времени.

Б) Система должна успевать реагировать на одновременно происходящие события. Даже если два или больше внешних событий происходят одновременно, система должна успеть среагировать на каждое из них в течение интервалов времени, критического для этих событий.

"Системой реального времени является такая система, корректность функционирования которой определяется не только корректностью выполнения вычислений, но и временем, в которое получен требуемый результат. Если требования по времени не выполняются, то считается, что произошел отказ системы"

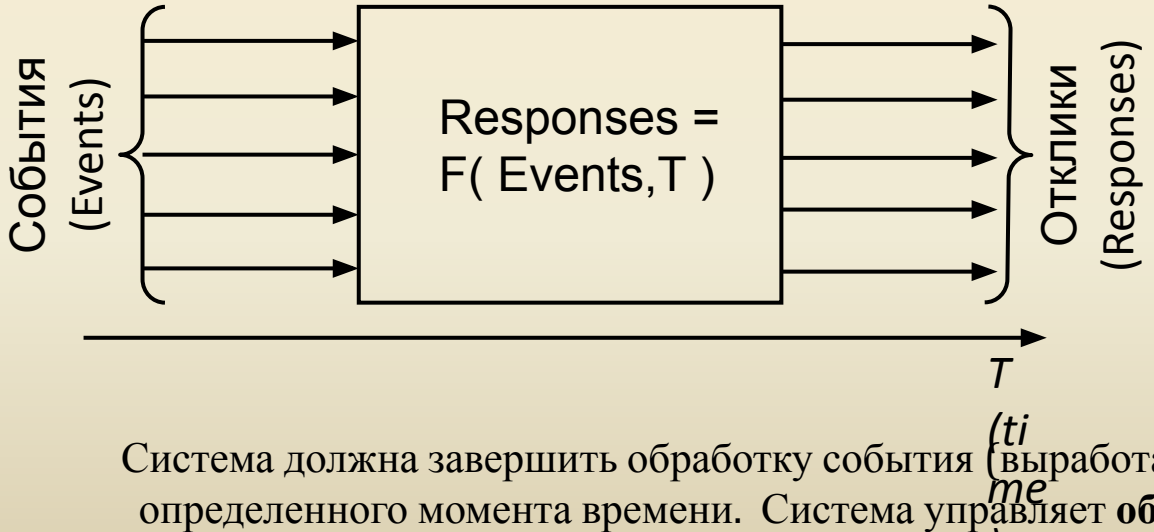
Для систем реального времени характерно следующее:

-- **гарантированное время реакции на внешние события** (например на прерывания от оборудования);

-- **жёсткая подсистема планирования процессов** (высокоприоритетные задачи не должны вытесняться низкоприоритетными, за некоторыми исключениями);

-- **повышенные требования к времени реакции на внешние события или реактивности** (задержка вызова обработчика прерывания не более десятков микросекунд, задержка при переключении задач не более сотен микросекунд).

Время реакции должно быть гарантированным.

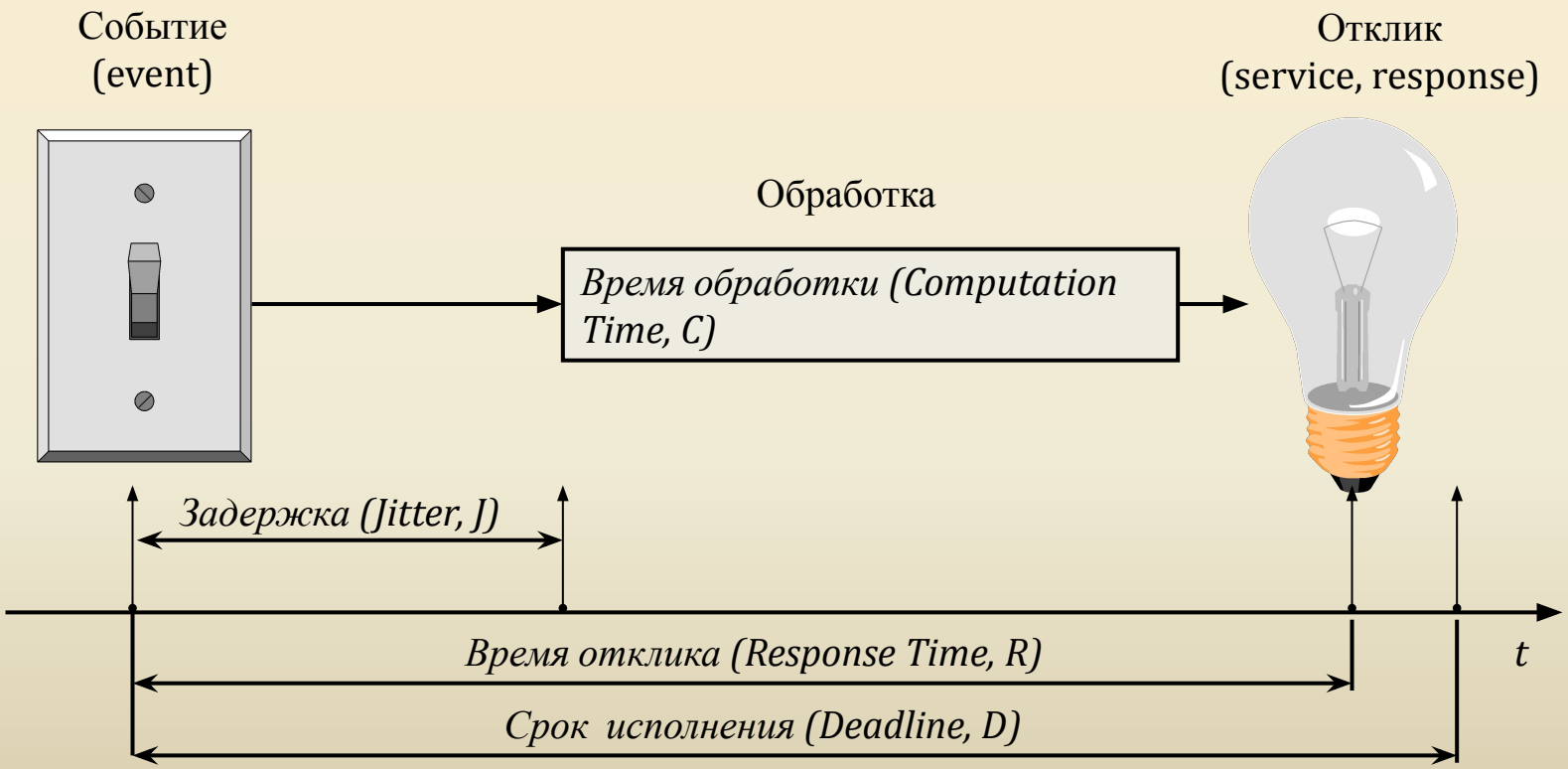


Система (приложение) реального времени - программная система, в которой корректность работы зависит не только от результатов вычислений, но также **от времени получения этих результатов.**

Система должна завершить обработку события (выработать отклик) не позднее заранее определенного момента времени. Система управляет **обработкой большого количества разных событий**.

-- Реальное время определяется соотношением срока исполнения и временем отклика.

-- Реальное время не зависит от того, “быстрая” система или “медленная” (то есть не зависит от единиц измерения времени).



Обработка “в реальном времени” означает “вовремя”

Вывод:

1. практически все системы промышленной автоматизации являются системами реального времени;

2. принадлежность системы к классу систем реального времени никак не связана с ее быстродействием.

Например, если ваша система предназначена для контроля уровня грунтовых вод, то даже выполняя измерения с периодичностью один раз за полчаса, она будет работать в реальном времени.

Например, Шахматная программа.

Быстродействие системы реального времени должно быть тем больше, чем больше скорость протекания процессов на объекте контроля и управления

§ 1.2 Типы СРВ

1. системы «жесткого» РВ

2. системы «мягкого» РВ

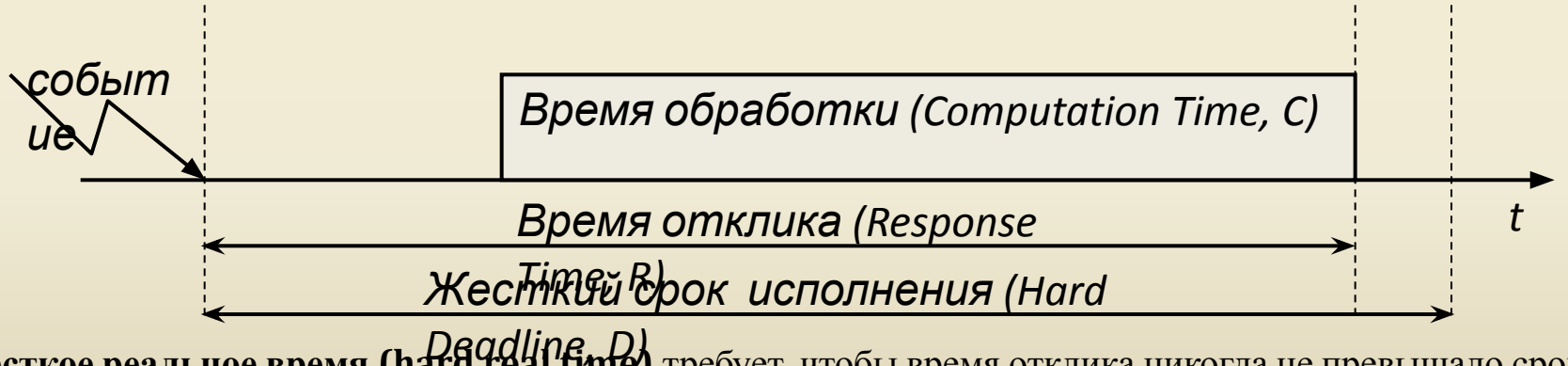
Системой «**жесткого**» реального времени называется система, где не способность обеспечить реакцию на какие-либо события в заданное время является отказом и ведет к невозможности решения поставленной задачи, т.е. системы жесткого реального времени не допускают никаких задержек реакции системы ни при каких условиях, так как:

-- результаты могут оказаться бесполезны в случае опоздания,

-- может произойти катастрофа в случае задержки реакции,

-- стоимость опоздания может оказаться бесконечно велика.

Пример: система управления двигателем; система торможения; подушки безопасности.



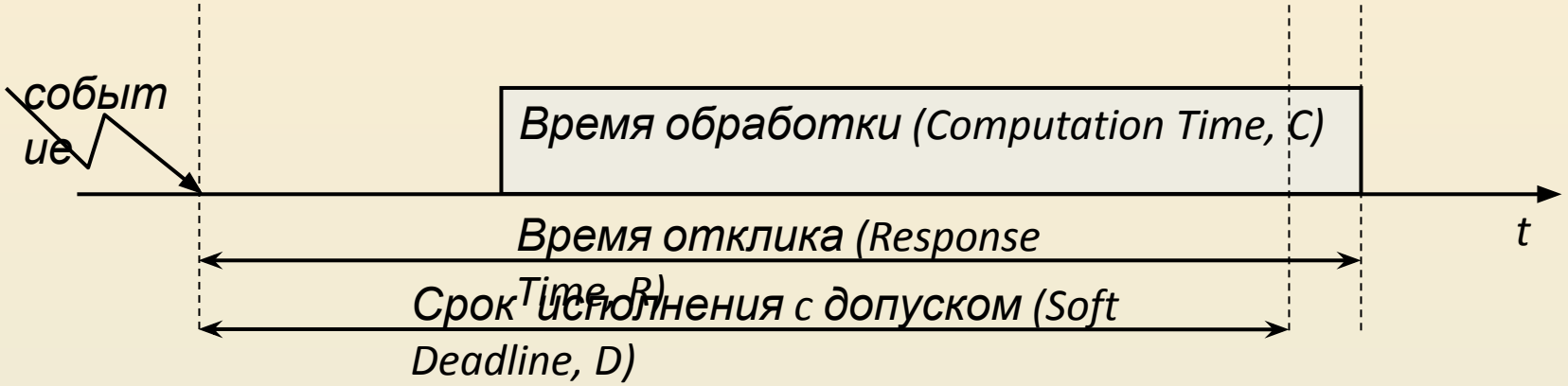
Жесткое реальное время (hard real time) требует, чтобы время отклика никогда не превышало срок исполнения (т.е. R **меньше либо равно** D).

В случае, если срок исполнения истекает, а отклик не был выработан, происходит **фатальный отказ системы**.

Требуется в большинстве встроенных приложений!

Системы «**мягкого**» реального времени характеризуются тем, что задержка реакции не критична, хотя и может привести к увеличению стоимости результатов и снижению производительности системы в целом.

«**deadline**» директивный срок, до истечения которого задача должна обязательно (для систем мягкого реального времени – желательно) выполниться.

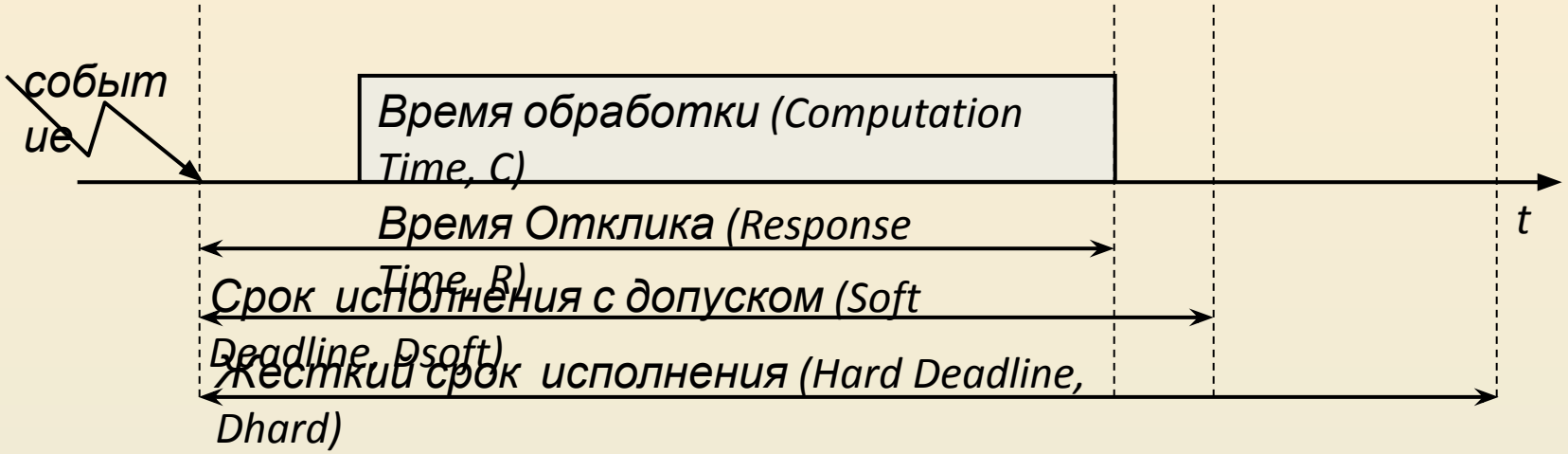


Реальное время с допусками (soft real time) допускает флуктуации времени отклика при условии, что среднее время отклика равно сроку исполнения (т.е. R **в среднем равно** D). Система работает хуже (деградирует), но **сохраняет работоспособность** даже если срок исполнения иногда просрочен.

Например: Сети передачи данных, сервер БД.

Основное отличие: система жесткого реального времени никогда не опоздает с реакцией на событие, система мягкого реального времени - не должна опаздывать с реакцией на событие.

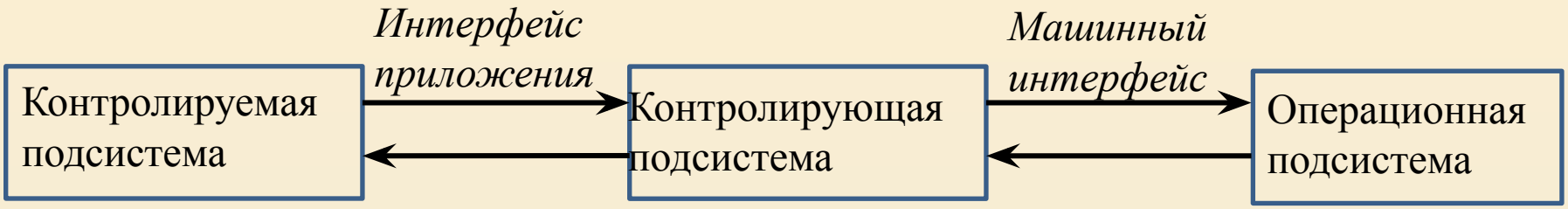
Комбинированное реальное время (firm real time) комбинирует два срока выполнения - короткого «с допуском» и более длинного «жесткого» (т.е. R **в среднем равно** Dsoft, но **меньше либо равно** Dhard).



Примеры:

- *мульти-медиа приложения*
- *высоко-скоростные сети передачи данных*

§ 1.3 Структура СРВ



Контролируемая (**Управляемая**) подсистема диктует требования в РВ, представлена задачами, которые используют оборудование управляемое подсистемой контроля.

Подсистема контроля (контролирующая) управляет некоторыми вычислениями и связью с оборудованием от управляемой подсистемы. Может быть построена из очень большого количества процессоров, управляющими такими местными ресурсами, как память и устройства хранения, доступ к локальной сети в РВ. Эти процессоры и ресурсы управляются системой ПО, которая называется ОС РВ (RTOS)

Подсистема оператора (операционная) контролирует полную деятельность системы.

Интерфейс приложения – датчики, приводы и т.д.

Машинный интерфейс связывает человека с машинной.

§ 1.4 Закрепление материала
Система автоматизации климатических испытаний

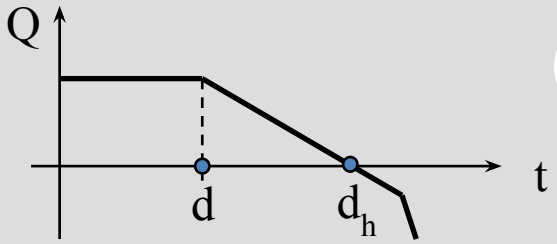
Функции системы:

1. Управление температурой, давлением, влажностью по заданной программе.
2. Контроль работоспособности оборудования
3. Диспетчеризация процесса испытаний.

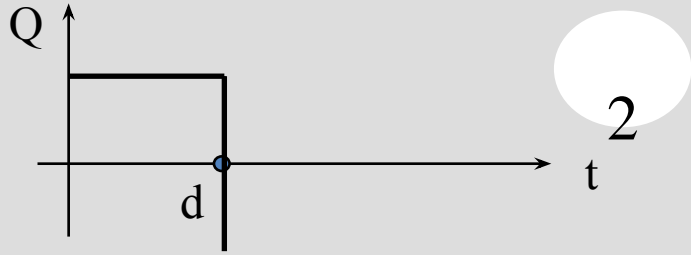
Какая из систем мягкого РВ, а какая жесткого???

Требования к времени реакции. *Q – «Значимость» результатов работы задачи*

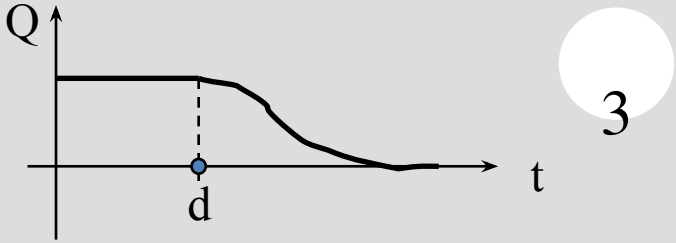
Управление температурой



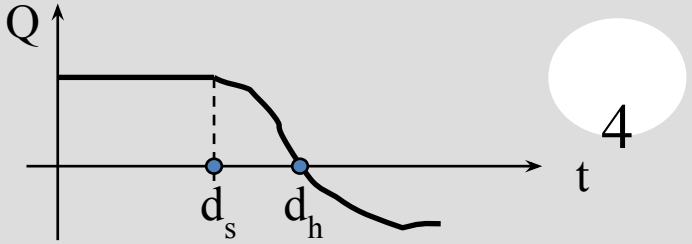
Контроль работоспособности



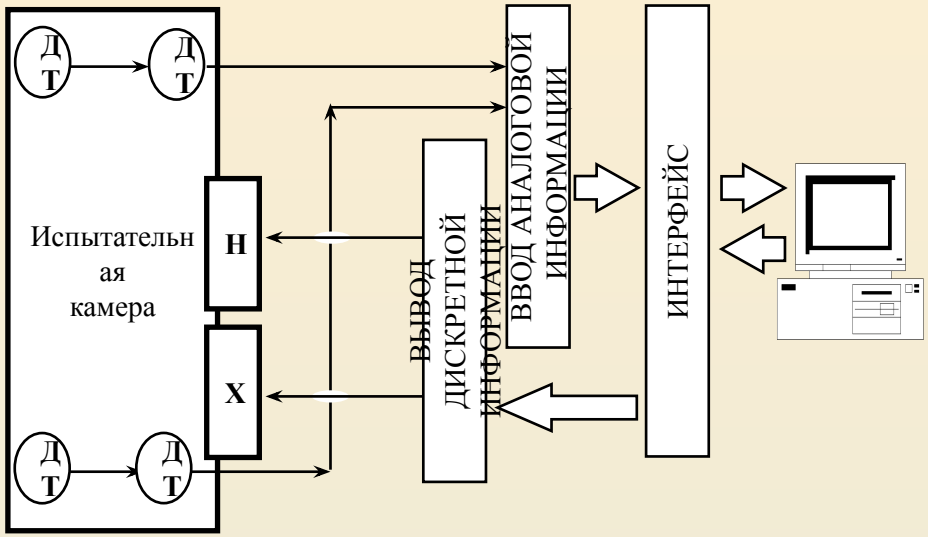
Диспетчеризация «а»
(отображение и регистрация)

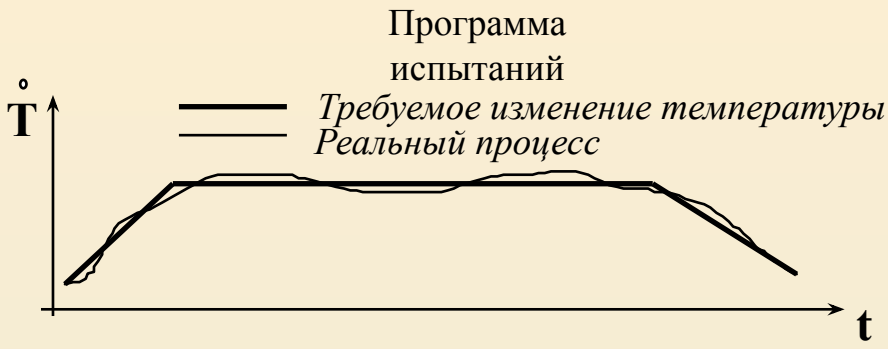


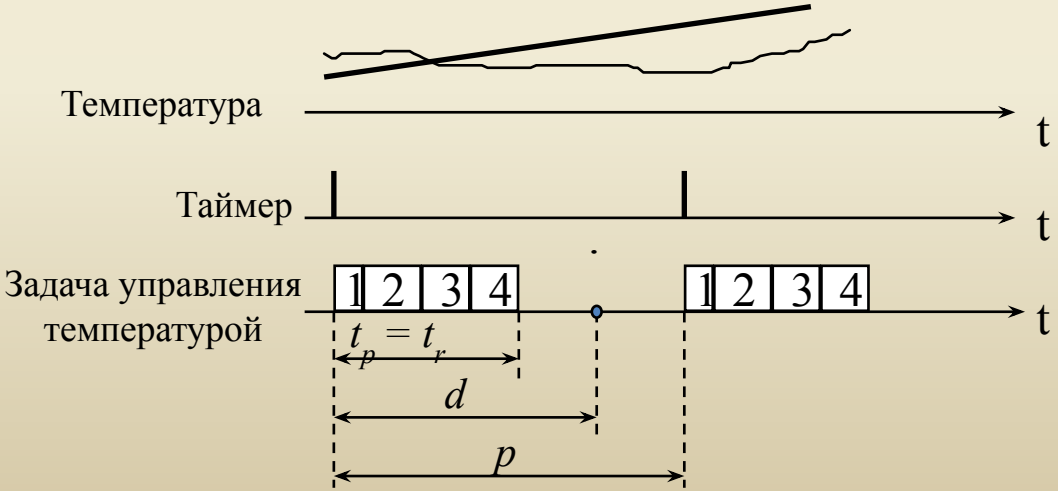
Диспетчеризация «в»
(интерпретация запросов)



1. Управление температурой, давлением, влажностью по заданной программе.







t_p - время выполнения (execution time, processing time)

t_r - время реакции системы (response time; в данном случае $t_p = t_r$)

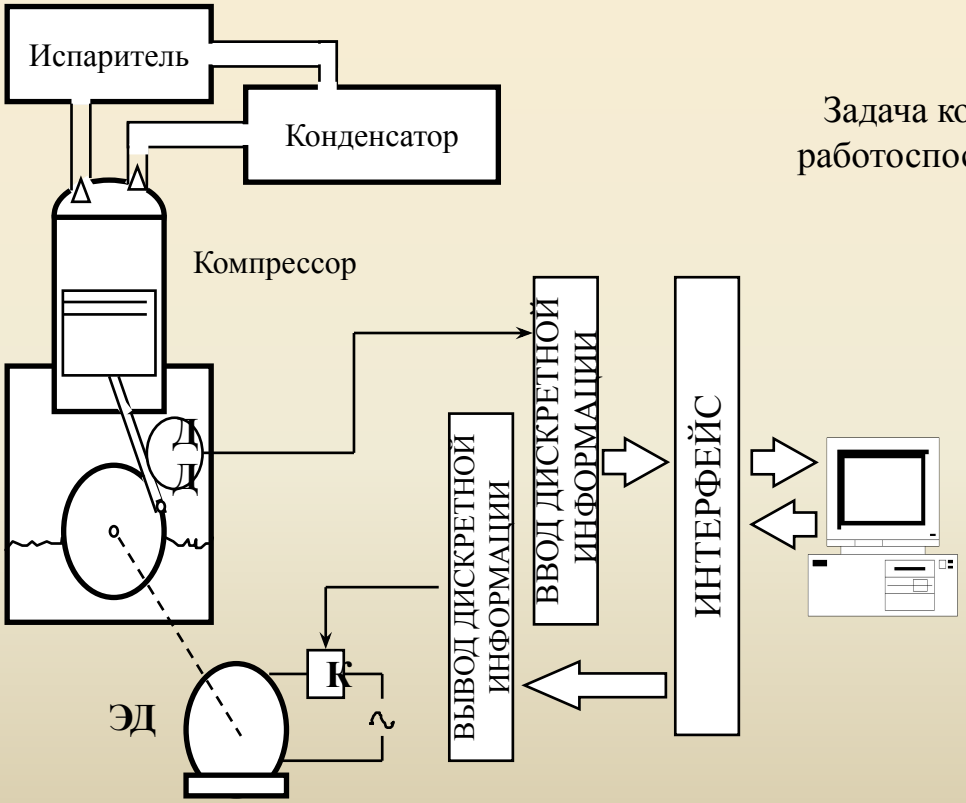
d - предельно допустимое время завершения (deadline)

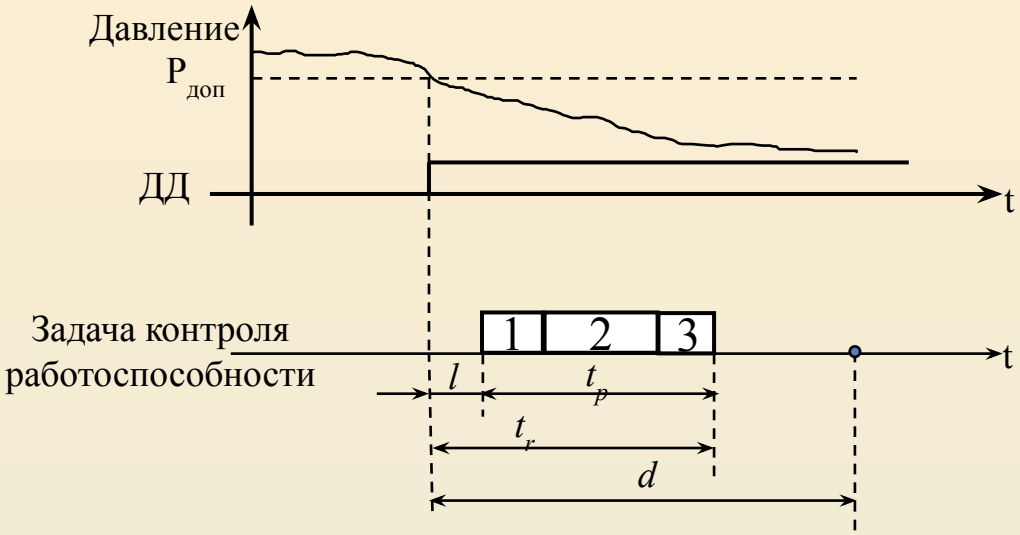
p - период активизации (period)

- *1 - измерение;*
- *2 - вычисление_программного_значения;*
- *3 - алгоритм_принятия_решения;*
- *4 - вывод_управляющего_воздействия;*

2. Задача контроля работоспособности испытательного оборудования

- *1 - процедура_обработки_события;*
- *2 - алгоритм принятия решения;*
- *3 - вывод_управляющего_воздействия;*





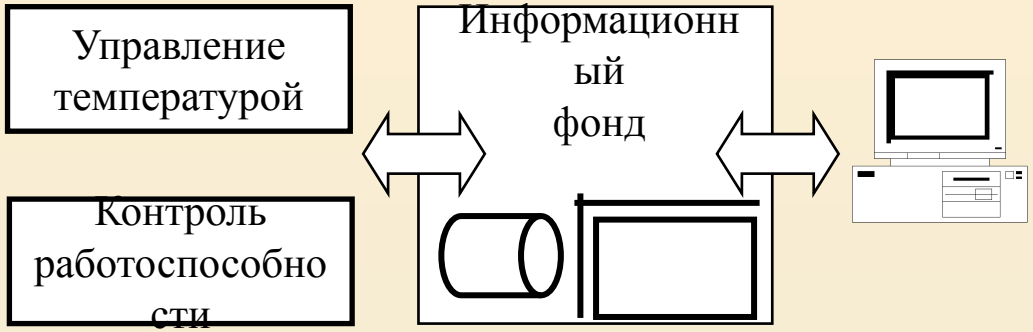
l - задержка выполнения процедуры обработки события (latency)

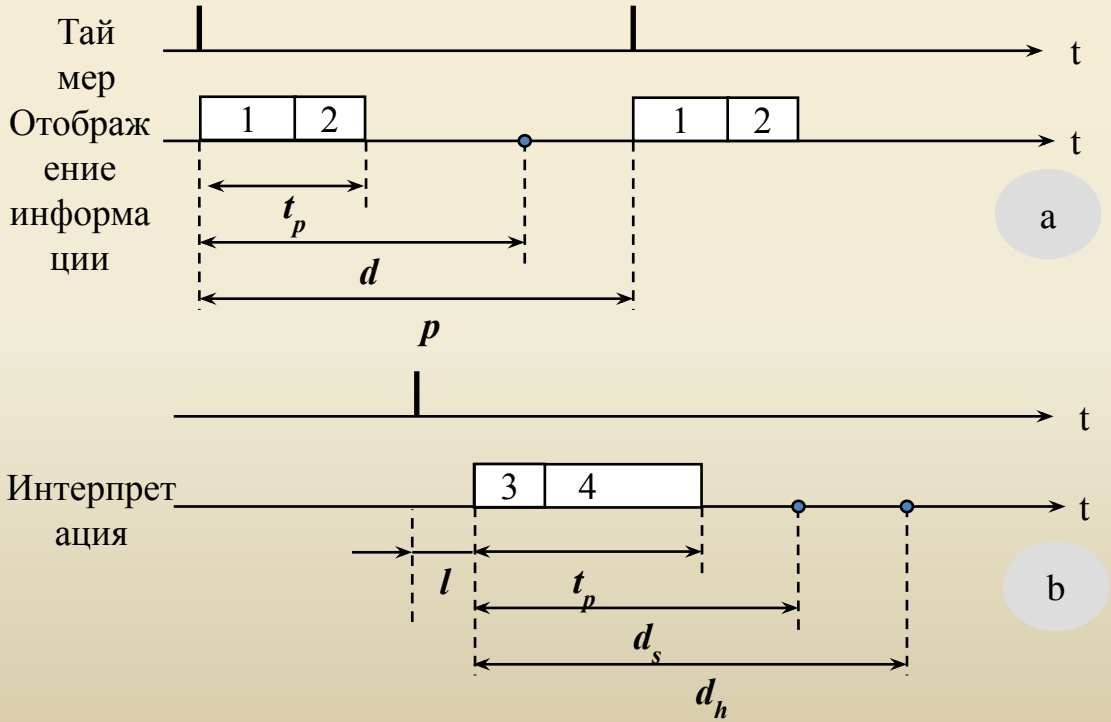
t_p - время выполнения (execution time, processing time)

t_r - время реакции системы (response time; в данном случае $t_p=t_r$)

d - предельно допустимое время завершения (deadline)

3. Задача диспетчеризации





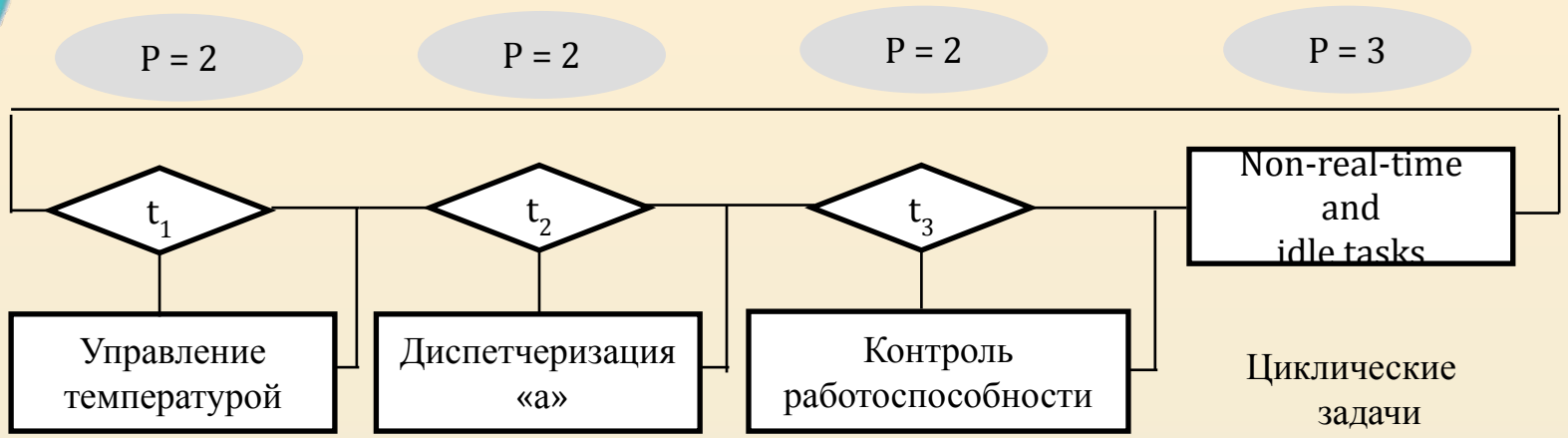
- *1 - обработка_данных;*
- *2 - вывод (экран, печать);*
- *3 - ввод_команды;*
- *4 - интерпретация_запроса;*

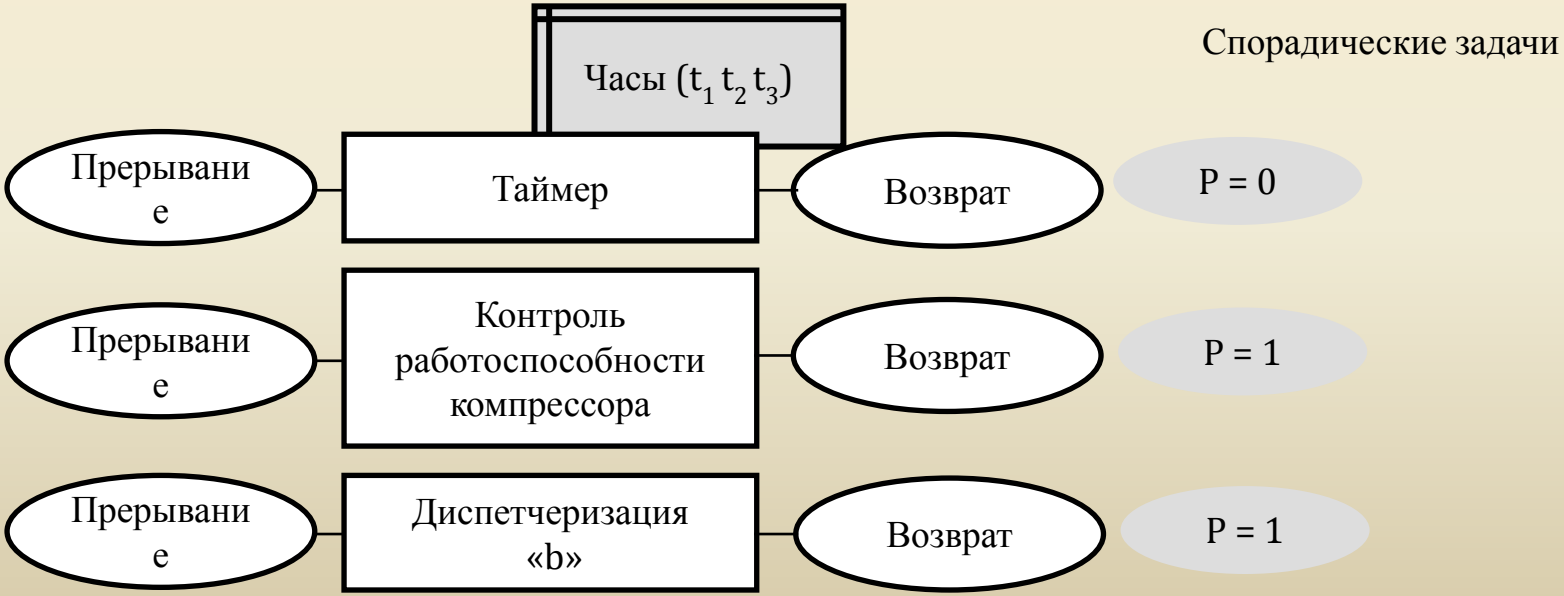
Требование к времени реакции

- ✔ Жесткое реальное время (hard real-time) - нарушения *deadline* приводит к аварийным (возможно - катастрофическим) последствиям и является недопустимым
- ✔ Мягкое реальное время (soft real-time) - нарушения *deadline* допускается (если такое событие происходит «*не слишком часто*» и *deadline* превышается на «*небольшой промежуток времени*»
- ✔ Комбинированное реальное время - определены два значения *deadline*, первое из которых – d_s (меньшее) - рассматривается как soft real-time ограничение, а второе - d_h (большее) – как hard real-time

Многозадачность!!

- ✔ СРВ - необходимо одновременно решать несколько задач, обслуживающих процесс внешней среды
- ✔ Задачи различают
  - ✔ по степени важности, присваивая им различные приоритеты
  - ✔ по способу их активизации во времени – периодические и спорадические (асинхронные)


P = 2
P = 2
P = 2
P = 3
t1
t2
t3
Non-real-time and idle tasks
Управление температурой
Диспетчеризация «a»
Контроль работоспособности
Циклические задачи


Спорадические задачи
Часы (t1 t2 t3)
Прерывание
Таймер
Возврат
P = 0
Прерывание
Контроль работоспособности компрессора
Возврат
P = 1
Прерывание
Диспетчеризация «b»
Возврат
P = 1

Особенности СРВ

- ✔ СРВ - система, *активно* взаимодействующая с внешней средой; временной масштаб процессов обработки информации в СРВ *согласуется* с временным масштабом процессов, протекающих во внешней среде.
- ✔ СРВ - многозадачная система; несколько задач выполняются *одновременно*, *обслуживая* процессы внешней среды. Различают циклические и спорадические задачи.
- ✔ Реальное время не означает «быстро», но «своевременно»; каждая задача должна закончится в пределах заданного для нее времени; в этом смысле говорят о «предсказуемости» (predictability) поведения системы
- ✔ Характеристики задачи реального времени:
  - длительность выполнения (execution time),
  - период активизации (period),
  - допустимое время завершения (deadline),
  - время выполнения в наихудшем случае (worst case execution time),
  - приоритет (priority),
  - время реакции (response time),
  - задержка выполнения процедуры обработки события (latency)

Особенности СРВ

- ✔ Каждая задача реального времени определяется постоянным набором процедур, повторяющихся в течении времени функционирования системы (от единиц до сотен и тысяч часов)

- ✔ По строгости требований на выполнения deadline задачи (системы) относят к классам «жесткого реального времени» (hard real-time), «мягкого реального времени» (soft real-time) и «комбинированного реального времени» (firm real-time)

- ✔ Наряду с задачами реального времени в системе решаются задачи «нереального времени» (non-real-time system), которым (как правило) определяется более низкий приоритет

- ✔ Повышенные требования к надежности по сравнению с системами «нереального времени»

§ 2 - Введение в ОС РВ

ОС РВ □ Системы жесткого РВ

2.1 **Требования к ОС РВ**:

-- ОС должна быть многозадачной и допускающей вытеснение,
-- ОС должна обладать понятием приоритета для потоков,
-- ОС должна поддерживать предсказуемые механизмы синхронизации,
-- ОС должна обеспечивать механизм наследования приоритетов,
-- поведение ОС должно быть известным и предсказуемым (задержки обработки прерываний, задержки переключения задач, задержки драйверов и т.д.); это значит, что во всех сценариях рабочей нагрузки системы должно быть определено максимальное время отклика.

Большинство ПО □ «мягкое» РВ.

Задача СРВ □ обеспечить уровень безопасного функционирование системы, даже если управляющая программа никогда не завершит работу.

2.2 Различия между ОС РВ и ОС общего назначения.

ОС общего назначения, ориентированы на оптимальное распределение ресурсов компьютера между пользователями и задачами (многопользовательские ОС, например UNIX).

В **ОС РВ** главной задачей является успеть среагировать на события, происходящие на объекте, все остальные задачи уходят на второй план.

Отличие - четкое разграничение систем разработки и систем исполнения.

Система исполнения в ОС РВ это набор инструментов (ядро, драйверы, исполняемые модули), обеспечивающих функционирование приложения реального времени. Система исполнения в ОС РВ и компьютер, на котором она исполняется называют «целевой» системой.

Система разработки - набор средств, обеспечивающих создание и отладку приложения реального времени (компиляторы, отладчики и т.д)

2.3 Характеристики ОС РВ

Временем реакции системы на события - интервал времени от события на объекте и до выполнения первой инструкции в программе обработки этого события является.

Время переключения контекста - время, которое система затрачивает на передачу управления от процесса к процессу. (Многозадачность)

Время перезагрузки системы – для систем, где требуется непрерывная работа. Устойчивость к перезагрузкам.

Вычислительное оборудование для СРВ:

- «обычные» компьютеры;
- промышленные компьютеры;
- встраиваемые системы.

Свойства ОС РВ:

1. Размер
2. Возможность исполнения системы из ПЗУ

Доп. Свойства ОС РВ:

1. Наличие необходимых драйверов устройств
2. Поддержка процессоров различной архитектуры
3. Специальный кроссплатформенный инструментарий разработчика.

2.4 Механизмы РВ

Механизмы РВ – важнейшие характеристики СРВ!!

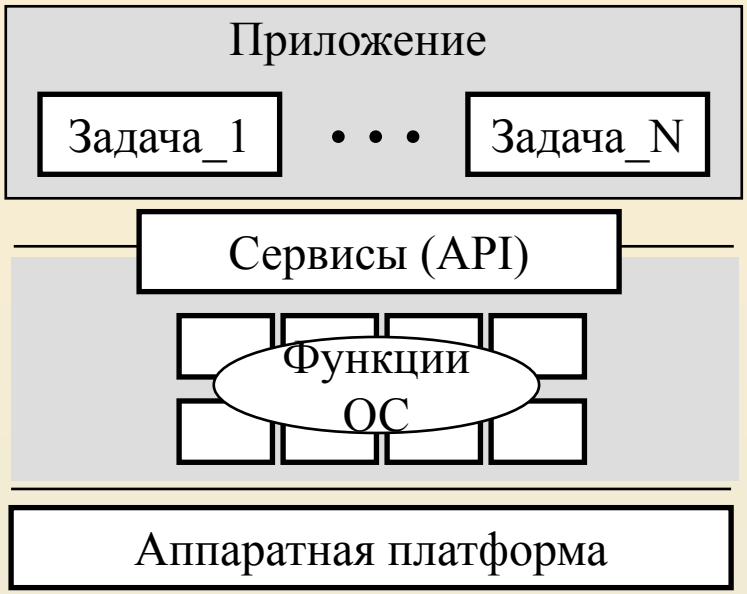
1. Системы приоритетов и алгоритмы диспетчеризации
2. Механизмы межзадачного взаимодействия (средства синхронизации процессов и передачи данных между ними: семафоры, мьютексы, события, сигналы, средства для работы с разделяемой памятью, каналы данных (pipes), очереди сообщений)
3. Средства для работы с таймерами (системы с жестким РВ),

позволяют:

- измерять и задавать различные промежутки времени (от 1 мкс),
- генерировать прерывания по истечении временных интервалов,
- создавать разовые и циклические будильники

2.5 Архитектура ОС РВ

ОС РВ с монолитной архитектурой



Характерно:
- жесткая связь между функциями ОС;
- плохая переносимость;
- сложность расширения;

- ✔ Прикладного уровня: состоит из работающих прикладных процессов;
- ✔ Системного уровня: состоит из монолитного ядра операционной системы, в котором можно выделить следующие части:

1. интерфейс между приложениями и ядром (API),
2. собственно ядро системы,
3. интерфейс между ядром и оборудованием (драйверы устройств).

ОС РВ с монолитной архитектурой

API в таких системах играет двойную роль:
- управление взаимодействием прикладных процессов и системы;
- обеспечение непрерывности выполнения кода системы

Преимущество монолитной архитектуры является относительная быстрота работы по сравнению с другими архитектурами.

Недостатки монолитной архитектуры:

- ✔ Системные вызовы, требующие переключения уровней привилегий должны реализовывать API как прерывания или ловушки. Это сильно увеличивает время их работы.
- ✔ Ядро не может быть прервано пользовательской задачей. Это может привести к тому, что высокоприоритетная задача может не получить управление из-за работы низко-приоритетной.
- ✔ Сложность переноса на новой архитектуры процессора из-за значительных ассемблерных вставок.
- ✔ Негибкость и сложность развития: изменение части ядра системы требует его полной перекомпиляции.

Модульная архитектура ОС РВ (на основе микроядра)

Попытка убрать узкое место – API и облегчить модернизацию системы и перенос ее на новые процессоры.
API обеспечивает связь прикладных процессов и специального модуля – менеджера процессов. Однако, теперь микроядро играет двойную роль

Недостатки модульной = монолитной.
Проблемы перешли с уровня API на уровень микроядра. Системный интерфейс по-прежнему не допускает переключения задач во время работы микроядра, только сократилось время пребывания в этом состоянии. API по-прежнему может быть реализован только на ассемблере, проблемы с переносимостью микроядра уменьшились (в связи с сокращением его размера), но остались.

Данное сочетание имеет ряд недостатков:
-В едином работающем комплексе (приложение + ОСРВ) разные компоненты используют разные подходы к разработке ПО.
-Не используются все возможности объектно-ориентированного подхода.
-Возникают некоторые потери производительности из-за разного типа интерфейсов в ОСРВ и приложении.
Естественно, возникает идея строить саму ОС РВ, используя объектно-ориентированный подход.

Объектно – ориентированная архитектура ОС РВ

API отсутствует!

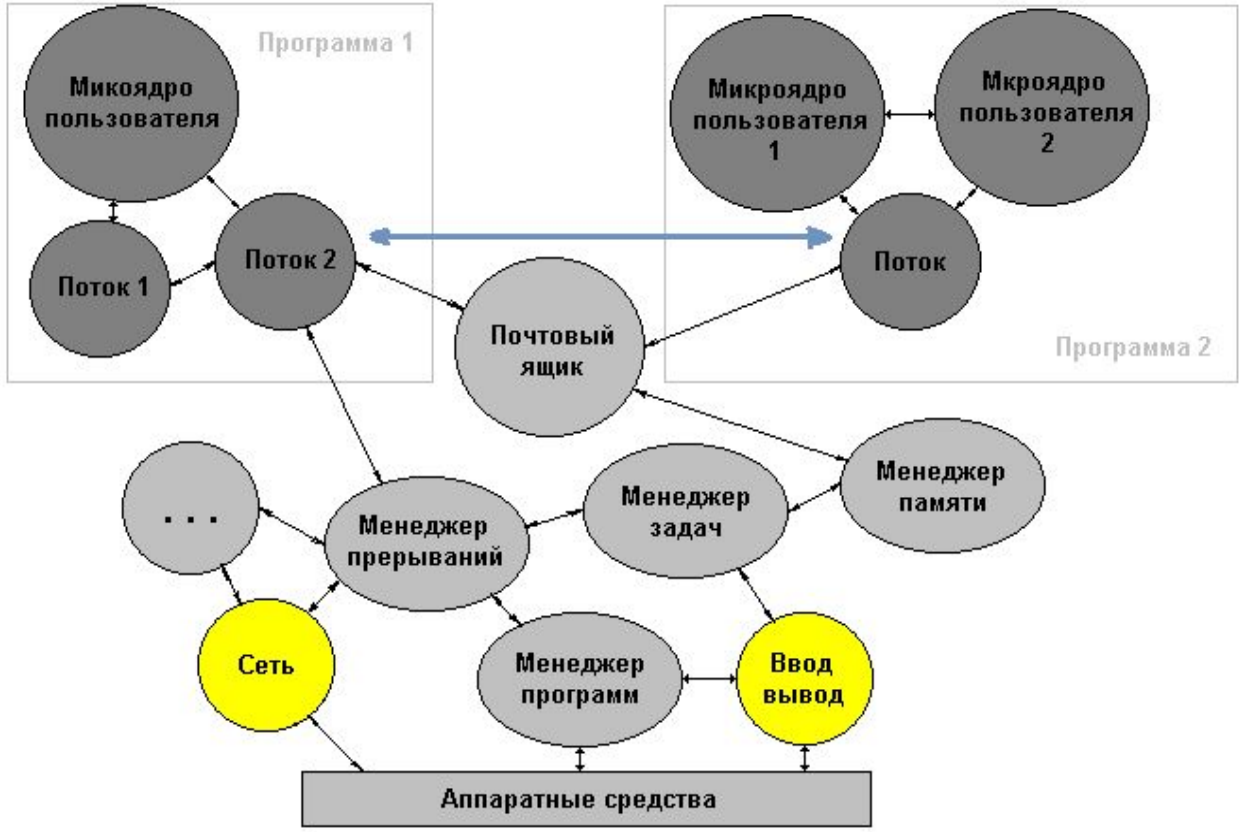
Взаимодействие между микроядрами с помощью вызова функций.

-> максимальная скорость системных вызовов.

Переключение задач в любое время.

Обеспечивает:

- Модульность.
- Легкость модернизации.
- Безопасность.



Роль API играет компилятор и динамический редактор объектных связей (linker). При старте linker загружает нужные ему микроядра (т.е. не все компоненты самой ОС должны быть загружены в ОП). Если микроядро уже загружено для другого приложения, оно повторно не загружается, а использует код и данные уже имеющегося ядра. Позволяет уменьшить объем требуемой памяти.

2.6 Классы СРВ

1. **Исполнительные СРВ –** программы для микропроцессоров; встраиваемые в устройства; малый объем -> язык низкого уровня (~Ассемблер).

   Достоинства: скорость и реактивность системы(потоки и мин. время переключения между ними)

   Недостатки: зависание всей системы при зависании потока.

2. **Минимальное ядро СРВ –** СРВ, обеспечивающие мин. среду исполнения. Только основные функции(нет управления памятью и диспетчер). Ядро – набор программ (типичные для встроенных систем низкого уровня функции). Разработка в инструментальной среде, выполнение – встроенные системы.
Системы – модульные, структурированные, компактны и предсказуемы. QNX
Достоинства – масштабируемость. На базе можно построить как компактные СРВ, так и системы серверного класса.

3. **Ядро СРВ и инструментальная среда –** ~~ ОС с полным сервисом.
Разработка в инструментальной среде, исполнение – на целевых системах.

4. **ОС с полным сервисом –** большие СРВ, + визуализация, + работа с БД, + доступ в Интернет

§ 3 Планирование задач и процессов ОС РВ

3.1 Строение ОС РВ

ОС РВ делят на 3 слоя:

- **Ядро –** содержит только необходимый минимум для работы системы: управление задачами, их синхронизация и взаимодействие; управление памятью и устройствами ввода/вывода; размер ядра очень ограничен: несколько килобайт.
- **Система управления –** содержит ядро и ряд дополнительных сервисов, расширяющих его возможности;
- **Система реального времени –** содержит систему управления и набор утилит: средства разработки (компиляторы, отладчики), средства визуализации.

3.2 Функции ядра ОС РВ

Ядро может обеспечивать сервис пяти типов:

1. **Синхронизация ресурсов.** Метод требует ограничить доступ к общим ресурсам. Двоичный семафор(избирательный доступ); Счетный семафор (для систем с большой ошибкоустойчивостью; одновременный доступ к ресурсу лишь определенному количеству процессов).

2. **Межзадачный обмен.** В случаях если,

- Для передачи данных между программами внутри одной и той же системы.
- Приложения взаимодействуют 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.

3. **Разделение данных.** Организация очереди данных.

4. **Обработка запросов внешних устройств.** Ядро должно обеспечивать службы ввода/вывода, позволяющие прикладным программам осуществлять чтение с этих устройств и запись на них.

5. **Обработка особых ситуаций.**

6. **Диспетчеризация (Планирование)!!!**

3.3 Процессы, потоки. Задачи.

Процесс - отдельно загружаемый программный модуль (файл), который, как правило, во время исполнения имеет в памяти свои независимые области для кода и данных, т.е. процесс – это

- Программа на стадии выполнения.
- «объект», которому выделено процессорное время.
- Асинхронная работа.

Состояния:

Выполнение — это активное состояние, во время которого процесс обладает всеми необходимыми ему ресурсами. В этом состоянии процесс непосредственно выполняется процессором.

Ожидание — это пассивное состояние, во время которого процесс заблокирован и не может быть выполнен, потому что ожидает какое-то событие, например, ввода данных или освобождения нужного ему устройства.

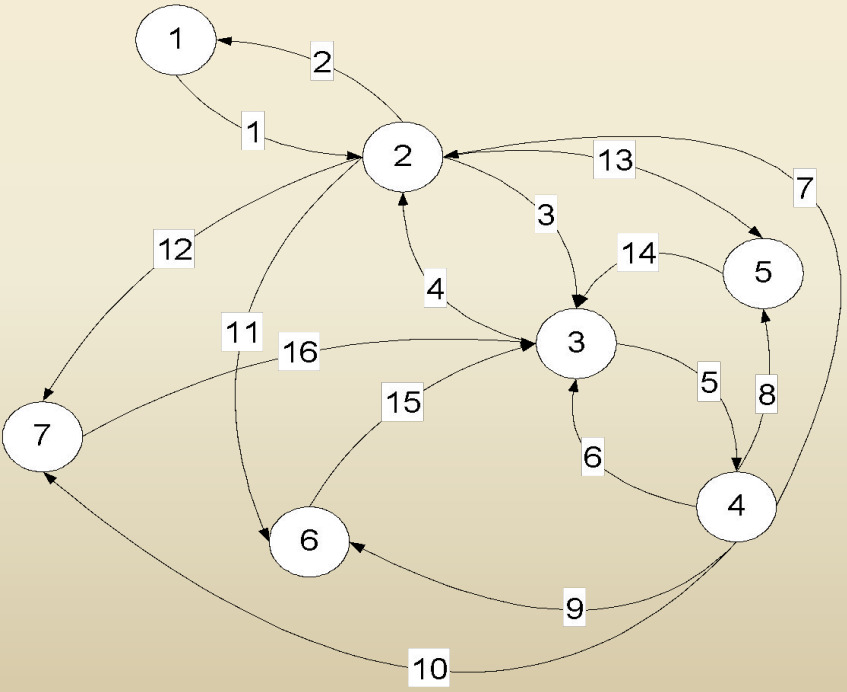
Готовность — это пассивное состояние, процесс тоже заблокирован, но в отличие от состояния ожидания, он заблокирован не по внутренним причинам, а по внешним, независящим от процесса, причинам.

Рождение процесса — это пассивное состояние, когда самого процесса еще нет, но уже готова структура для появления процесса.

Смерть процесса — самого процесса уже нет, но может случиться, что его «место", то есть структура данных , осталась в списке процессов.

Состояния процесса:

1. не существует
2. не обслуживается
3. готов
4. выполняется
5. ожидает ресурс
6. ожидает назначенное время
7. ожидает события



Переходы:

1. переход 1-2 создание процесса
2. переход 2-1 уничтожение процесса
3. переход 2-3 активизация процесса диспетчером
4. переход 3-2 деактивизация процесса
5. переход 3-4 загрузка на выполнение процесса диспетчером
6. переход 4-3 требование обслуживания от процессора другим процессом (preemption – приоритетное переключение)
7. переход 4-2 завершение процесса
8. переход 4-5 блокировка процесса до освобождения требуемого ресурса
9. переход 4-6 блокировка процесса до истечения заданного времени
10. переход 4-7 блокировка процесса до прихода события
11. переход 2-6 активизация процесса приводит к ожиданию временной задержки
12. переход 2-7 активизация процесса приводит к ожиданию события
13. переход 2-5 активизация процесса приводит к ожиданию освобождения ресурса
14. переход 5-3 активизация процесса из-за освобождения ожидавшегося ресурса
15. переход 6-3 активизация процесса по истечении заданного времени
16. переход 7-3 активизация процесса из-за прихода ожидавшегося события

Создание процесса состоит из присвоения новому процессу идентификатора процесса и подготовки информации, которая определяет окружение процесса.

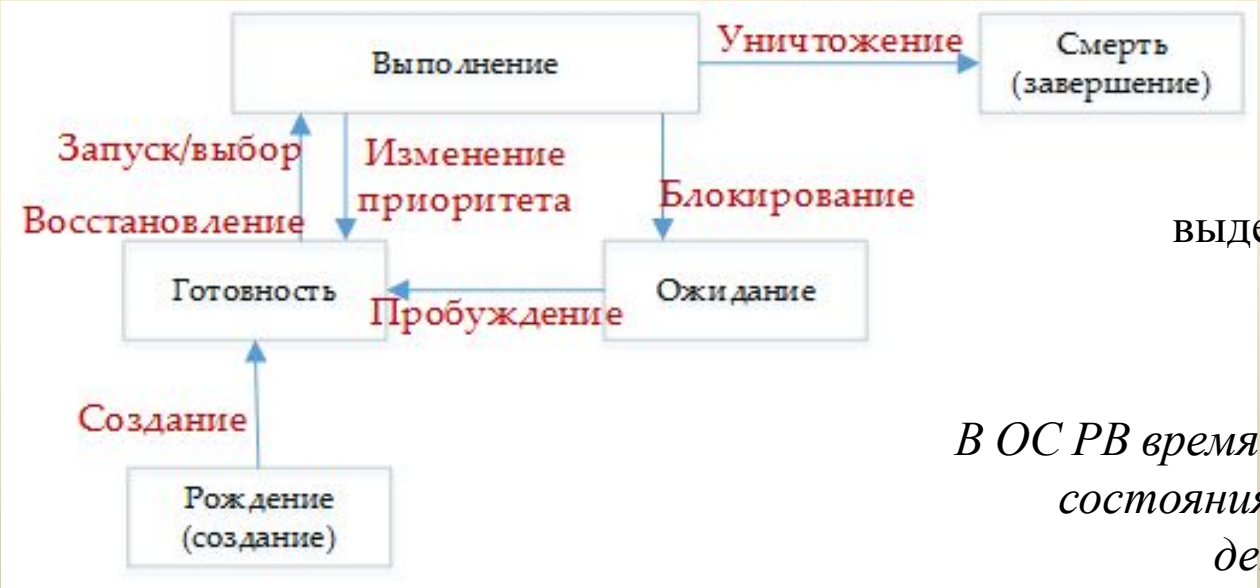
жизненный цикл процесса:

1. создание
2. загрузка
3. выполнение
4. завершение.

Загрузка процесса означает загрузку в память кода процесса.

После процесс готов *к **выполнению***. Он начинает конкурировать с другими процессами за ресурсы процессора. Процесс может выполняться, а может блокироваться по тем или иным причинам.

Операции над процессами:



Завершение процесса освобождение всех ресурсов, выделенных процессу (файловых дескрипторов, памяти и т.д.)

В ОС РВ время перехода процесса из одного состояния в другое должно быть детерминировано.

Для **создания** процесса ОС нужно:

1. Присвоить процессу имя
2. Добавить информацию о процессе в список процессов
3. Определить приоритет процесса
4. Сформировать блок управления процессом
5. Предоставить процессу нужные ему ресурсы

Иерархия процессов:

Процесс не может взяться из ниоткуда: его обязательно должен запустить какой-то процесс.

Процесс, запущенный другим процессом, называется дочерним (child) процессом или потомком.

Процесс, который запустил новый процесс называется родительским (parent), родителем или просто — предком.

У каждого процесса есть два атрибута — PID (Process ID) - идентификатор процесса и PPID (Parent Process ID) — идентификатор родительского процесса.

Процессы создают иерархию в виде дерева. Самым «главным» предком, то есть процессом, стоящим на вершине этого дерева, является процесс init (PID=1).

Потоки

Поток (thread) - подпроцесс, или легковесный процесс *(light-weight process)*, выполняющийся в контексте полноценного процесса.

С помощью процессов можно организовать параллельное выполнение программ. Для этого процессы клонируются, а затем между ними организуется взаимодействие (ресурсо-затратный способ).

С другой стороны, для организации параллельного выполнения и взаимодействия процессов можно использовать механизм ***многопоточности****. Основной единицей здесь является поток, который представляет собой облегченную версию процесса.*

Преимущества потоков перед процессами.

Им требуется:

--меньше времени для создания нового потока, поскольку создаваемый поток использует адресное пространство текущего процесса;

--меньше времени для завершения потока;

--меньше времени для переключения между двумя потоками в пределах процесса;

--меньше коммуникационных расходов, поскольку потоки разделяют все ресурсы, и в частности адресное пространство. Данные, продуцируемые одним из потоков, немедленно становятся доступными всем другим потокам.

Преимущества потоков:

1. Так как множество потоков способно размещаться внутри одного *EXE*-модуля, это позволяет экономить ресурсы как внешней, так и внутренней памяти.

2. Использование потоками общей области памяти позволяет эффективно организовать межзадачный обмен сообщениями. Процессы не имеют общей области памяти. Поэтому ОС должна либо целиком скопировать сообщение из области памяти одной задачи в область памяти другой, либо предусмотреть специальные механизмы, которые позволили бы одной задаче получить доступ к сообщению из области памяти другой задачи.

3. Как правило, контекст потоков меньше, чем контекст процессов, а значит, время переключения между задачами-потоками меньше, чем между задачами-процессами.

4. Так как все потоки, а иногда и само ядро РВ размещаются в одном *EXE*-модуле, значительно упрощается использование программ-отладчиков (*debugger*).

Недостатки потоков:

1. Потоки не могут быть подгружены динамически.

Чтобы добавить новый поток, необходимо провести соответствующие изменения в исходных текстах и перекомпилировать приложение. Процессы, в отличие от потоков, подгружаемы, что позволяет динамически изменять функции системы в процессе ее работы. Кроме того, так как процессам соответствуют отдельные программные модули, они могут быть разработаны различными компаниями, чем достигается дополнительная гибкость и возможность использования ранее наработанного ПО.

2. То, что потоки имеют доступ к областям данных друг друга, может привести к ситуации, когда некорректно работающий поток способен испортить данные другого потока. В отличие от этого процессы защищены от взаимного влияния, а попытка записи в «не свою» память приводит, как правило, к возникновению специального прерывания по обработке «исключительных ситуаций». Реализация механизмов управления процессами и потоками, возможность их взаимного сосуществования и взаимодействия определяются конкретным ПО РВ.

*Если операционная система поддерживает концепции потоков в рамках одного процесса, она называется **многопоточной**.*

Преимущества многопоточности:

- ✔ **Улучшенная реакция приложения,** *любая программа, содержащая много не зависящих друг от друга действий, может быть перепроектирована так, чтобы каждое действие выполнялось в отдельном потоке.*
- ✔ **Более эффективное использование мультипроцессирования,** *как правило, приложения, реализующие параллелизм через потоки, не должны учитывать число доступных процессоров. Производительность приложения равномерно увеличивается при наличии дополнительных процессоров.*
- ✔ **Улучшенная структура программы,** *некоторые программы более эффективно представляются в виде нескольких независимых или полуавтономных единиц, чем в виде единой монолитной программы. Многопоточные программы легче адаптировать к изменениям требований пользователя.*
- ✔ **Эффективное использование ресурсов системы,** *программы, использующие два или более процессов, которые имеют доступ к общим данным через разделяемую память, содержат более одного потока управления. При этом каждый процесс имеет полное адресное пространство и состояние в операционной системе. Стоимость создания и поддержания большого количества служебной информации делает каждый процесс более затратным, чем поток. Кроме того, разделение работы между процессами может потребовать от программиста значительных усилий, чтобы обеспечить связь между потоками в различных процессах или синхронизировать их действия.*

СРВ — это программно-аппаратный комплекс, осуществляющий мониторинг какого-то объекта и/или управление им в условиях временных ограничений. Возникающие на объекте события подлежат обработке в СРВ.
Каждому типу события сопоставляется задача.

Задача -блок программного кода, ответственный за обработку тех или иных событий, возникающих на объекте управления.

Может быть представлена в виде:

- ✔ Отдельного процесса.
- ✔ Потока управления внутри процесса (нити, легковесного процесса).
- ✔ Обработчика прерывания/подпрограммы.

Работа задачи - процесс исполнения блока программного кода в ходе обработки события.

Каждая работа задачи характеризуется следующими временными параметрами:

r (Release Time) — момент времени, когда задача становится готовой к исполнению (например, процесс переходит в состояние готовности)
d (Absolute Deadline) — абсолютный крайний срок, момент времени, к которому задача должна завершить очередную работу.
s (Start Time) — момент времени, когда задача начала исполняться на процессоре
c (Complition Time) — момент времени, когда задача закончила работу, обработав событие
D (Relative Deadline) — относительный крайний срок. $D = d - r$
e (Execution Time) — время исполнения задачи при выполнении ею очередной работы. $e = c - s$
R (Response Time) — время отклика. $R = c - r$

Параметры определяются как:

- ✔ Времена перехода задач в состояние готовности определяются природой управляемого объекта.
- ✔ Крайние сроки определяет разработчик СРВ, исходя из свойств управляемого объекта.
- ✔ Времена исполнения задач определяются архитектурой процессора, его тактовой частотой и конкретной реализацией того или иного алгоритма. (Для определения величины можно использовать 2 подхода)

2. подсчет количества тактов процессора, необходимых на выполнение той или иной задачи.

такой подсчет чрезвычайно усложняется в случае, если процессор содержит механизмы типа конвейеров и всевозможных кэшей.

2. времена исполнения непосредственно измеряются.

в случае процессоров с конвейерами и кэшами такие измерения не дают гарантии, что будет измерено именно максимальное время исполнения того или иного кода.

Наконец, системы, использующие механизмы подкачки страниц, также являются менее предсказуемыми и поэтому считается, что такого рода механизмы являются «врагами» систем реального времени. Недаром в различного рода стандартах, касающихся СРВ, предусмотрены средства блокировки страниц памяти.

Свойства задач.

*Вся важная, информация о задаче хранится в унифицированной структуре данных – **управляющем блоке** (Task Control Block, TCB). В блоке хранятся такие параметры, как имя и номер задачи, верхняя и нижняя границы стека, ссылка на очередь сообщений, статус задачи, приоритет и т. п.*

Приоритет – это некое целое число, присваиваемое задаче и характеризующее ее важность по сравнению с другими задачами, выполняемыми в системе.

Приоритет используется в основном планировщиком задач для определения того, какая из готовых к работе задач должна получить управление.

системы с динамической приоритетностью

приоритет задач может меняться в процессе исполнения

системы со статической приоритетностью

приоритет задач жестко задается на этапе разработки или во время начального конфигурирования системы

Контекст задачи – это набор данных, содержащий всю необходимую информацию для возобновления выполнения задачи с того места, где она была ранее прервана.

Часто контекст хранится в TCB и включает в себя такие данные, как счетчик команд, указатель стека, регистры CPU и FPU и т. п.

Планировщик зада[illegible] и восстанавливает контекст задачи, назначенной к исполнению.

Такое переключение контекстов и является **основным механизмом** ОС РВ при переходе от выполнения одной задачи к выполнению другой.

Состояние (статус) задачи. С точки зрения ОС, задача может находиться в нескольких состояниях. Число и название этих состояний различаются от одной ОС к другой.

1. ***Активная задача*** *– это задача, выполняемая системой в текущий момент времени.*
2. ***Готовая задача*** *– это задача, готовая к выполнению и ожидающая у планировщика своей «очереди».*
3. ***Блокированная задача*** *– это задача, выполнение которой приостановлено до наступления определенных событий. Такими событиями могут быть освобождение необходимого задаче ресурса, поступление ожидаемого сообщения, завершение интервала ожидания и т. п.*

Пустая задача (Idle Task) – это задача, запускаемая самой операционной системой в момент инициализации и выполняемая только тогда, когда в системе нег других готовых для выполнения задач.

Пустая задача запускается с самым низким приоритетом и, как правило, представляет собой бесконечный цикл «ничего не делать». Наличие пустой задачи предоставляет операционной системе удобный механизм отработки ситуаций, когда нет ни одной готовой к выполнению задачи.

Многократный запуск задач. Многозадачные ОС позволяют запускать несколько копий одной и той же задачи. При этом для каждой такой копии создается свой *TCB* и выделяется своя область памяти. В целях экономии памяти может быть предусмотрено совместное использование одного и того же исполняемого кода для всех запущенных копий. В этом случае программа должна обеспечивать повторную входимость (реентерабельность). Кроме того, программа не должна использовать временные файлы с фиксированными именами и должна корректно осуществлять доступ к глобальным ресурсам.

Реентерабельность означает возможность без негативных последствий временно прервать выполнение какой-либо функции или подпрограммы, а затем вызвать эту функцию или подпрограмму снова. Частным проявлением реентерабельности является рекурсия, когда тело подпрограммы содержит вызов самой себя. Классическим примером нереентерабельной системы является *DOS*, а типичной причиной нереентерабельности служит использование глобальных переменных.

Выводы.

Задачи реального времени могут выполняться в виде процессов или потоков. Между ними не должно быть слишком много взаимодействий, и в большинстве случаев они имеют различную природу — жесткого реального времени, мягкого реального времени, условного времени. Для обеспечения более эффективного решения задач реального времени стоит использовать многопотоковые программы как менее требовательные к ресурсам.

Windows NT как ОС РВ.

7.1.2. Управление ресурсами в ОСРВ

7.1.2.1. Управление процессами
Управление локальными ресурсами. Управление процессами.
Контекст и дескриптор процесса.
Алгоритмы планирования процессов. Вытесняющие и невытесняющие алгоритмы планирования.
Алгоритмы планирования процессов. Абсолютные и относительные приоритеты.
Средства синхронизации и взаимодействия процессов. Критические секции.
Средства синхронизации и взаимодействия процессов. Тупики.

7.1.2.2. Управление памятью
Управление памятью в ОС РВ. Методы распределения памяти без использования дискового пространства.
Управление памятью в ОС РВ. Методы распределения памяти с использованием дискового пространства.
Управление памятью в ОС РВ. Свопинг.
Принцип кэширования данных.

7.1.3. QNX

7.1.3.1. Архитектура
Архитектура и функции ядра.
Виды IPC. Связь между процессами посредством сообщений.
Виды IPC. Связь между процессами посредством сигналов.
Виды IPC. Связь между процессами посредством Proxy.
Особенности построения системы.
Микроядро. Связь между процессами.

7.1.3.2. Методы планирования
Методы планирования процессов.
Приоритет, управляемый клиентом.