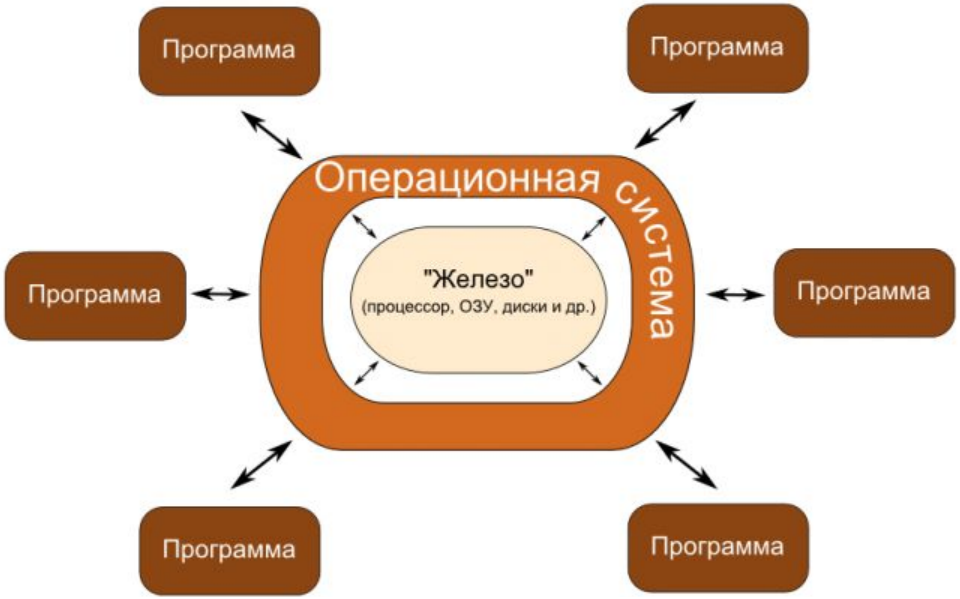
# Операционные системы реального времени

Ч1

Операционная система, ОС (англ. operating system, OS) — комплекс взаимосвязанных программ, предназначенных для управления ресурсами вычислительного устройства и организации взаимодействия с пользователем.

В логической структуре типичной вычислительной системы операционная система занимает положение между техническими устройствами — с одной стороны — и прикладными программами с другой.




Под реальным временем понимается количественная характеристика, которая может быть измерена реальными физическими часами, в отличие от **логического времени**, определяющего лишь качественную характеристику, выражаемую относительным порядком следования событий. Говорят, что система работает в режиме реального времени, если для описания работы этой системы требуются количественные *временны́е* характеристики.

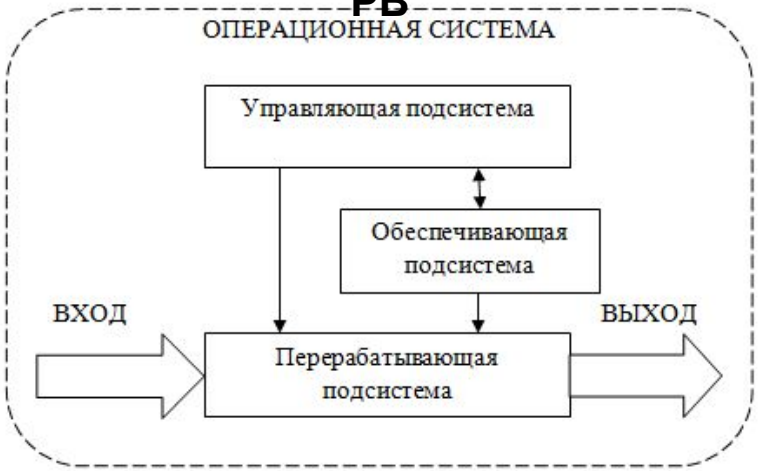
Операционная система реального времени, ОС РВ (англ. real-time operating system, RTOS) — тип операционной системы, основное назначение которой — предоставление необходимого и достаточного набора функций, обеспечивающих работу программных средств систем реального времени на конкретном аппаратном оборудовании.

| Задача ОС | ОС общего назначения | ОС реального времени |
|---|---|---|
| **Основная** | Оптимально распределить ресурсы компьютера между пользователями и задачами | Успеть среагировать на события, происходящие на оборудовании |
| **На что ориентирована** | Обработка действий пользователя | Обработка внешних событий |
| **Как позиционируется** | Воспринимается пользователем как набор приложений, готовых к использованию | Инструмент для создания конкретного аппаратно-программного комплекса реального времени |
| **Кому предназначена** | Пользователь средней квалификации | Квалифицированный разработчик |

Для систем реального времени характерно следующее:

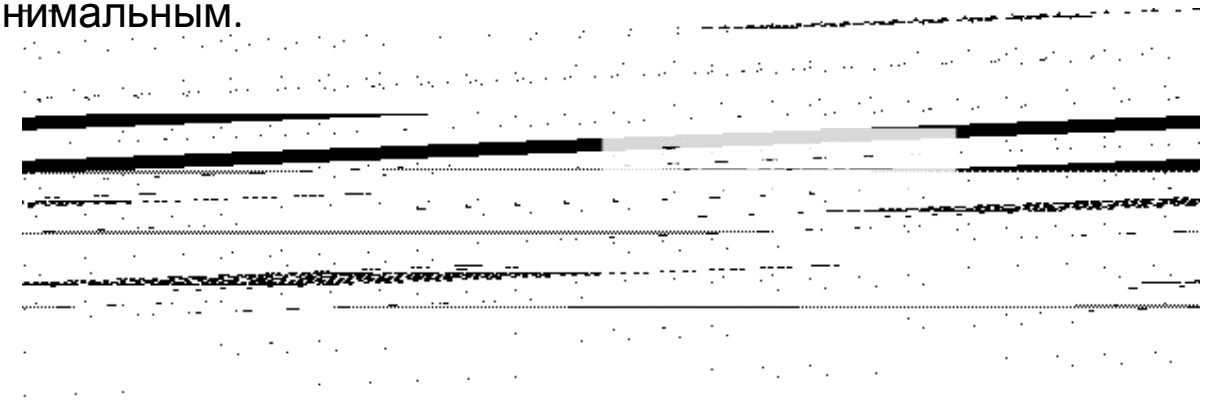
- гарантированное время реакции на внешние события (например на прерывания от оборудования);
- жёсткая подсистема планирования процессов (высокоприоритетные задачи не должны вытесняться низкоприоритетными, за некоторыми исключениями);

# Виды ОС РВ

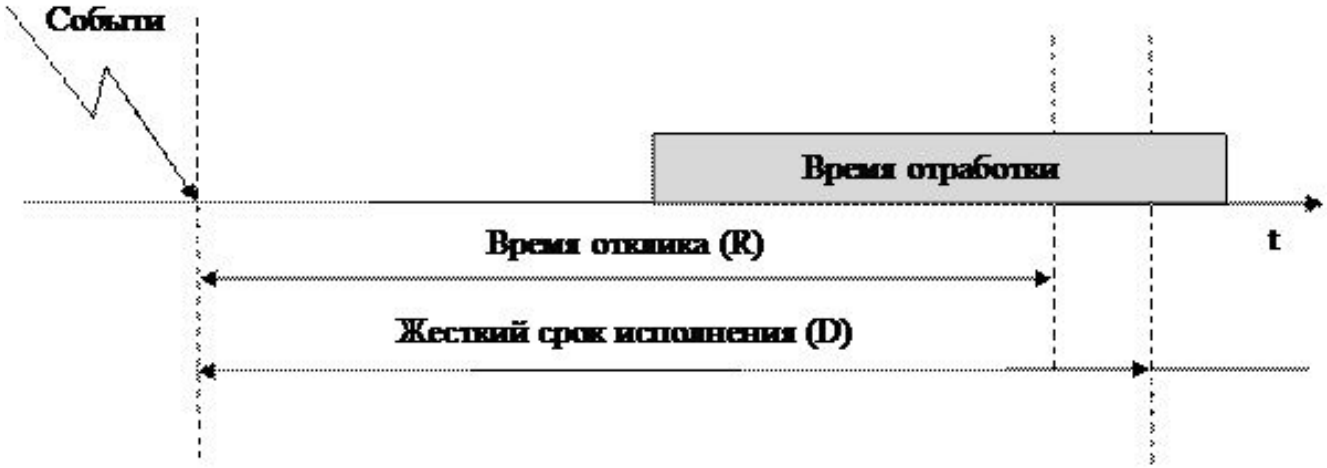



Динамические свойства программ реального времени принято характеризовать тремя определениями:

**Жесткое реальное время.** Предусматривает наличие гарантированного времени отклика системы на конкретное событие, например, аппаратное прерывание, выдачу команды управления и тому подобное. Абсолютная величина времени отклика большого значения не имеет и зависит от объекта управления. При практическом применении время реакции должно быть минимальным.



**Мягкое реальное время.** В этом случае ожидающееся время отклика системы является величиной скорее индикативной, нежели директивной. Предполагается что в большинстве случаев отклик уложится в заданные пределы, но и задержка в реакции системы некатастрофична.




**Интерактивное реальное время**. Является скорее психологической, нежели технической характеристикой. Определяет время, в течение которого оператор-человек способен спокойно, без нервозности, ожидать реакции системы на данные им указания.

# Последовательное программирование

**Программа** – это описание объектов (констант и переменных) и операций, совершаемых над ними. Программа – это чистая информация.

Наиболее распространенный способ создания программ – это **последовательное программирование.** Оно подразумевает, что операторы (команды) программы выполняются в определённой, заранее известной последовательности (последовательный алгоритм).

**Целью последовательной программы** является **преобразование входных данных**, заданных в определенной форме, **в выходные данные**, имеющие другую форму, **в соответствии с некоторым алгоритмом** – методом решения.


Рис.1. Обработка данных последовательной программой

**Последовательная программа** работает как фильтр для исходных данных.

Результат (выходные данные) полностью определяются входными данными и алгоритмом их обработки (программа). Временные показатели достижения результата выполнения алгоритма играют второстепенную роль.

**Результат не зависит** ни **от инструментальных** (определяют усилия и время, затраченные на разработку и характеристики исполняемого кода) ни **аппаратных средств** (определяют скорость выполнения программы).**В любом случае результаты (выходные данные) будут одинаковыми.**

# Параллельное программирование

**Параллельный алгоритм** — алгоритм, который может быть реализован по частям на одном или на множестве различных вычислительных устройств связанных вычислительной сетью с последующим объединением полученных результатов и получением корректного результата.

**Параллельное программирование** − **независимые программные модули** или задачи, которые выполняются (активны) одновременно, то есть работают параллельно, при этом каждая задача выполняет свои специфические функции, а отдельные программные модули взаимодействуют между собой.

Отличительные этапы параллельного программирования:

- Выявление параллелизма: анализ задачи с целью выделить подзадачи, которые могут выполняться одновременно.
- Определение параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять подзадачи. Для этого часто требуется найти зависимости между подзадачами и организовать исходный код так, чтобы ими можно было эффективно управлять.
- Выражение параллелизма: реализация параллельного алгоритма в исходном коде с помощью системы обозначений параллельного программирования.

В большинстве случаев применение обычных приемов последовательного программирования не позволяет построить систему реального времени.

**Программирование для системы реального времени** сильно отличается от последовательного программирования. Необходимо постоянно иметь в виду условия (окружающую среду), в которой работает программа.

В *системах реального времени* (СРВ) внешние сигналы, как правило, требуют немедленной (или фиксированной по времени) реакции процессора.

Одна из наиболее важных особенностей СРВ является **конкретное, чётко определённое время реакции** на входные сигналы, которое должно удовлетворять заданным ограничениям.

**Время реакции системы на внешние события**

ОСРВ должна обеспечить требуемый уровень сервиса в заданный интервал времени. Этот интервал времени задается обычно периодичностью и скоростью процессов, которым управляет система и не зависит от разработчика программы.

Время реакции системы на «событие» – **от** «события» на объекте и **до** первой реакции программы обработки на это событие и является требуемым интервалом времени.
Проектируя систему реального времени, необходимо уметь вычислять этот интервал.
Специальные требования к программированию в реальном времени, в частности необходимость быстро реагировать на внешние запросы, нельзя адекватно реализовать с помощью обычных приемов последовательного программирования.

Для систем реального времени наиболее подходят и, поэтому, чаще всего используются, именно методы параллельного программирования.

# Программы и процессы

**Основным объектом в СРВ** является **задача, процесс (*task, process*)**.
**Программы** – это информация о том, как обрабатывать и преобразовывать исходные данные, а **Задачи** – состоят из процессов исполнения программы процессором**.**
**Процесс** – это блок программного кода, ответственный за обработку тех или иных событий, возникающих на объекте управления.
С помощью процессов можно организовать параллельное выполнение программ.
Программа, написанная на одном и том же языке высокого уровня, а затем откомпилированная и исполняемая на разных ЭВМ, порождает различные процессы, каждый из которых имеет собственные:

1. **области кода** – инструкций программы;
2. **области данных** – переменные и константы;
3. **heap** (куча) – область свободной, динамически распределяемой рабочей памяти;
4. **стек –** область памяти которая примыкает к куче и расширяется за ее счёт.


Рис.2. Организация внутренней памяти процесса

Процесс не может перейти из одного состояния в другое самостоятельно. Изменением состояния процессов занимается ОС, совершая операции над ними.

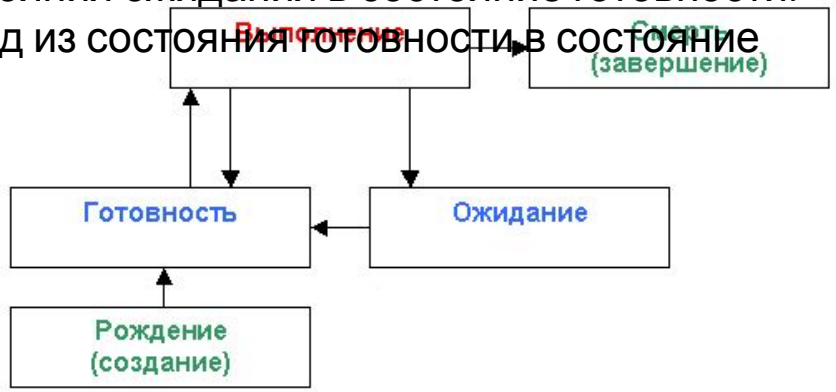
## Операции над процессами

Над процессами можно производить следующие операции:

- Создание процесса — это переход из состояния рождения в состояние готовности.
- Уничтожение процесса — это переход из состояния выполнения в состояние завершения.
- Восстановление процесса — переход из состояния готовности в состояние выполнения.
- Изменение приоритета процесса — переход из выполнения в готовность.
- Блокирование процесса — переход в состояние ожидания из состояния выполнения.
- Пробуждение процесса — переход из состояния ожидания в состояние готовности.
- Запуск процесса (или его выбор) — переход из состояния готовности в состояние выполнения.

Операции можно объединить в три пары:

- Создание процесса – завершение процесса;
- Приостановка процесса – запуск процесса;
- Блокировка процесса – разблокирование процесса.



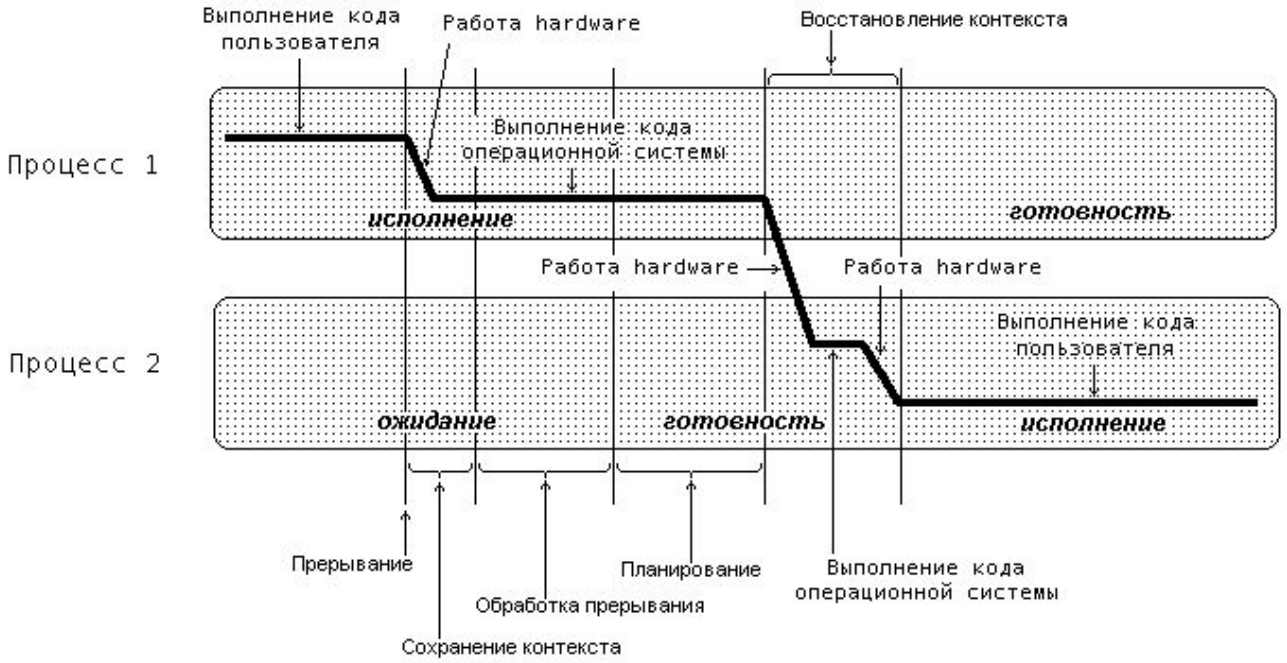

- Операции создания и завершения процесса являются одноразовыми.
- Все остальные операции, связанные с изменением состояния процессов являются многоразовыми.

Для создания процесса операционной системе нужно:

- Присвоить процессу имя.
- Добавить информацию о процессе в список процессов.
- Определить приоритет процесса.
- Сформировать блок управления процессом.
- Предоставить процессу нужные ему ресурсы.

Каждый процесс в любой момент времени находится в точно определенном состоянии, однозначно описываемом некоторой базовой информацией (например, содержимое регистров процессора, расположение областей кода, данных и стека и так далее) называемой – контекстом.

Существование контекста – это общее свойство процессов, а то, какие регистры, структуры и указатели реально являются частью контекста, зависит от используемого процессора и операцион



Процессы формируются из ***потоков*** (*thread*).
**Поток** – это подпроцесс, или «легковесный» процесс (*light-weight process*), выполняющийся в контексте полноценного процесса.

**Потоки** − частный тип процесса, представляют собой часть программы, которая может независимо исполняться на том же самом или другом процессоре.

Потоки порождаются процессом и выполняются параллельно, то есть поток − это дочерний процесс**.**
В отличие от процессов, потоки имеют доступ к пространству памяти и ресурсам своего процесса. Но каждый поток имеет собственный *контекст***,** то есть имеет свой стек исполнения и некоторый объём статической памяти для локальных переменных.

Для организации параллельного выполнения и взаимодействия процессов используется механизм многопоточности. Основной единицей многопоточности является поток, который представляет собой облегченную версию процесса.

Владельцу вычислительного ресурса, обычно называемому процессом или задачей, присущи:

- виртуальное адресное пространство;
- индивидуальный доступ к процессору, другим процессам, файлам, и ресурсам ввода — вывода.

В рамках процесса поток обычно является модулем для диспетчеризации. Ему обычно присущи следующие состояния:

- состояние выполнения (активное, готовность и так далее);
- сохранение контекста потока в неактивном состоянии.

# Преимущества многопоточности

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

Многопоточные приложения имеют ряд преимуществ:

- **Улучшенная реакция приложения** — любая программа, содержащая много не зависящих друг от друга действий, может быть перепроектирована так, чтобы каждое действие выполнялось в отдельном потоке. Например, пользователь многопоточного интерфейса не должен ждать завершения одной задачи, чтобы начать выполнение другой.
- **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. Численные алгоритмы и приложения с высокой степенью параллелизма, например перемножение матриц, могут выполняться намного быстрее.
- **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

# Многозадачность

**Многозада́чность** (англ. *multitasking*) — свойство операционной системы или среды выполнения обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких задач. Истинная многозадачность операционной системы возможна только в распределённых вычислительных системах.

Существует 2 типа многозадачности:

- *Процессная многозадачность* (основанная на процессах — одновременно выполняющихся по сути подпрограммах). Здесь подпрограмма — наименьший элемент управляемого кода, которым может управлять планировщик операционной системы.
- *Поточная многозадачность* (основанная на потоках). Наименьший элемент управляемого кода — поток.

Все процессы (потоки) совместно используют вычислительные ресурсы системы, но более или менее независимы друг от друга.

Примитивные многозадачные среды обеспечивают чистое «разделение ресурсов», когда за каждой задачей закрепляется определённый участок памяти, и задача активизируется в строго определённые интервалы времени.

Более развитые многозадачные системы проводят распределение ресурсов динамически, когда задача стартует в памяти или покидает память в зависимости от её приоритета и от стратегии системы.

**Для параллельных процессов конечный результат однозначно предсказать нельзя** – задачи выполняются, по крайней мере с внешней точки зрения, в случайной последовательности.

**Программирование в реальном времени** требует одновременного исполнения нескольких процессов на одной ЭВМ.

# Функции операционных систем в среде реального времени

**Операционная система** − это сложный программный продукт, предназначенный для управления аппаратными и программными ресурсами вычислительной системы.

Операционная система предоставляет каждому **процессу** «виртуальную (логическую) среду», базирующуюся на понятии *вычислительного ресурса*, который включает в себя:

- время процессора и
- объём памяти.

**«Виртуальная среда»** работает как прослойка между рабочим приложением и операционной системой, что позволяет избежать конфликтов между различными приложениями. Ее характеристики могут как совпадать, так и не совпадать с параметрами реального (физически имеющегося) оборудования.
Назначение многозадачного режима для обычных операционных систем и для ОСРВ существенно различаются между собой.

- Для “обычной” ОС многозадачный режим использует механизм разделения времени (многопользовательский режим), для того, чтобы обеспечить одновременный доступ нескольких пользователей (приложений) к относительно дорогим (дефицитным) вычислительным ресурсам и, соответственно, разделить между ними эксплуатационные расходы, то есть повысить экономическую эффективность оборудования.
- Для ОСРВ многозадачный режим служит для изоляции друг от друга разных операций и распределение рабочий нагрузки между отдельными программными модулями оптимальным образом. Единственным "пользователем" ОСРВ в этом случае является управляемая система.

# Управление процессором и состояния процесса

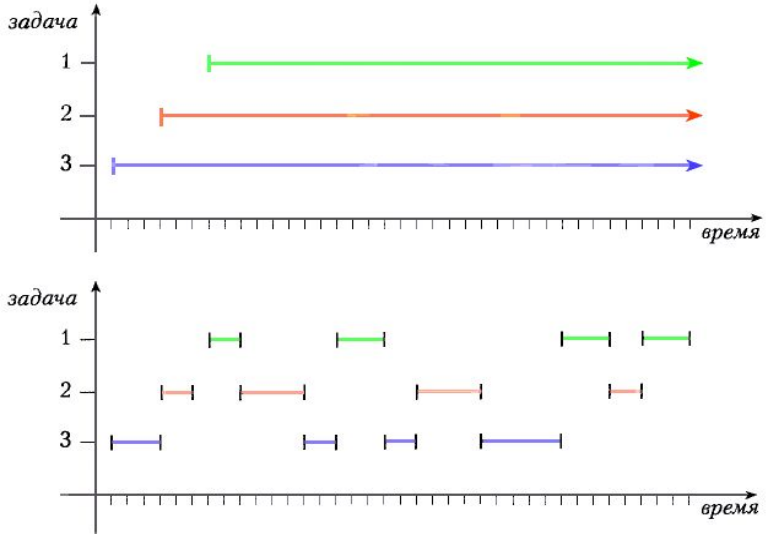
Основными объектами в многозадачной среде являются задачи или процессы, описываемые своим контекстом.

Реально на одном процессоре в любой момент времени может исполняться только один процесс, так как процессор принципиально является последовательным устройством.

- процесс должен быть завершен за время отпущенного ему периода (срок исполнения),
- процессы не зависят друг от друга,
- каждому процессу выделяется одинаковое процессорное время на каждом интервале,
- у непериодических процессов нет жестких сроков исполнения,
- прерывание процесса происходит за ограниченное время.

Контекст исполняемого процесса всегда можно **"заморозить"**, сохранив содержимое регистров процессора в специальной области памяти — стеке.

При остановке текущего процесса процессор продолжает исполнение других процессов.




Внешний эффект одновременного исполнения нескольких процессов , с точки зрения пользователя

Реальное распределение времени процессора

Рис. 4. Принцип организации многозадачного режима.

Операции по переключению процессов критичны по времени и должны осуществляться с максимальной эффективностью.

На процессорах, первоначально не разработанных для многозадачного режима, переключение процессов – процедура сохранения и восстановления контекста – реализуется длинной последовательностью стандартных инструкций процессора.

В набор команд процессора, спроектированного для работы в многозадачном режиме, входят **специальные инструкции для сохранения и восстановления контекста.**

Если в очереди активных (готовых) задач имеется более одной задачи, то появляется необходимость в планировании исполнения задач.

Алгоритм планирования задач является основным отличием систем реального времени от "обычных" операционных систем.

- В "обычных" ОС целью планирования является обеспечение выполнения всех задач из очереди готовых задач, обеспечивая иллюзию их параллельной работы и не допуская монополизацию процессора какой-либо из задач.
- В ОСРВ же целью планирования является обеспечение выполнения каждой готовой задачи к определенному моменту времени, при этом часто "параллельность" работы задач не допускается, поскольку тогда время исполнения задачи будет зависеть от наличия других задач.

Важнейшим требованием при планировании задач в ОСРВ является предсказуемость времени работы (исполнения) задачи.

Время исполнения задачи не должно зависеть от текущей загруженности системы, количества задач в очередях ожидания (процессора) поступления новых событий и так далее.

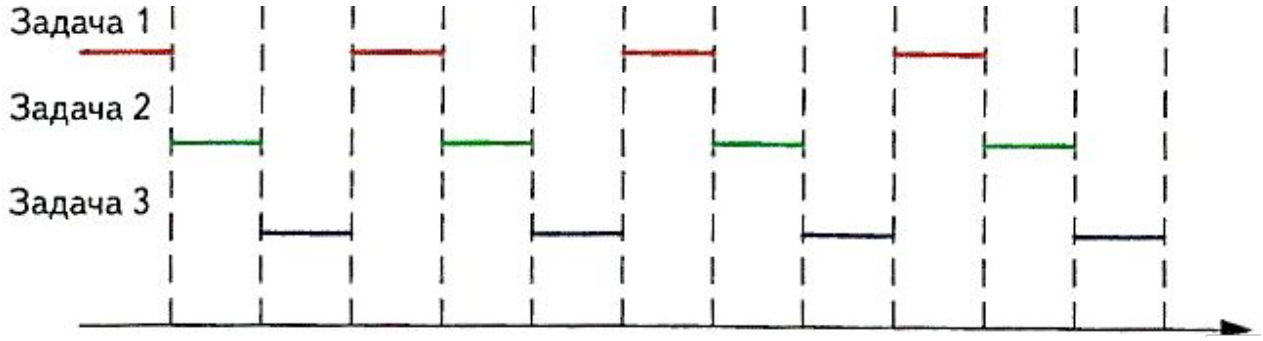
При этом желательно, чтобы длина этих очередей не была бы ограничена (то есть ограничена только объемом памяти, доступной системе).

Важнейшей функцией ядра ОСРВ является **диспетчеризация** (планирование).

**Планировщик задач** (scheduler) — это модуль (программа), отвечающий за разделение времени имеющихся процессоров между выполняющимися задачами.

**Планировщик** должен определять, какому процессу должно быть передано управление, а также должен определить время, выделяемое каждому процессу.

Планировщик отвечает за переключение задач из состояния блокировки в состояние готовности, и за выбор задачи из числа готовых для исполнения процессором.




Каждый раз, когда планировщик задач получает сигнал о наступлении некоторого внешнего события (триггер), причина которого может быть как аппаратная, так и программная, он действует по следующему алгоритму:

1. определяет, должна ли текущая выполняемая задача продолжать работать;
2. устанавливает последовательность исполнения задач;
3. сохраняет контекст остановленной задачи (чтобы она потом возобновила работу с места остановки);
4. загружает контекст для следующей задачи;
5. запускает эту задачу.

Ключевым вопросом планирования является выбор момента принятия решения:

1. Когда создается новый процесс, необходимо решить, какой процесс запустить, родительский или дочерний. Поскольку оба процесса находятся в состоянии готовности, эта ситуация не выходит за рамки обычного и планировщик может запустить любой из двух процессов.
2. Когда процесс завершает работу. Так как текущий процесс уже не существует, то необходимо из набора готовых процессов выбрать и запустить следующий.
3. Когда процесс блокируется по какой-либо причине, необходимо выбрать и запустить другой процесс.
4. Когда процесс уже разблокирован требуется принятие решения по диспетчеризации.
5. Когда истёк квант времени отпущенный процессу.