Распределенные информационные системы

Взаимодействие приложений. Многозадачные ОС. Процессы. Потоки

Вопросы

- Возникновение многозадачности
- Процессы:
  - Общие понятия
  - Состояния
- Потоки
- Файберы
- POSIX
- Межпроцессное взаимодействие – IPC

Последовательное выполнение

Переключение задач

Параллельное выполнение

ИСТОРИЯ МНОГОЗАДАЧНЫХ ОС

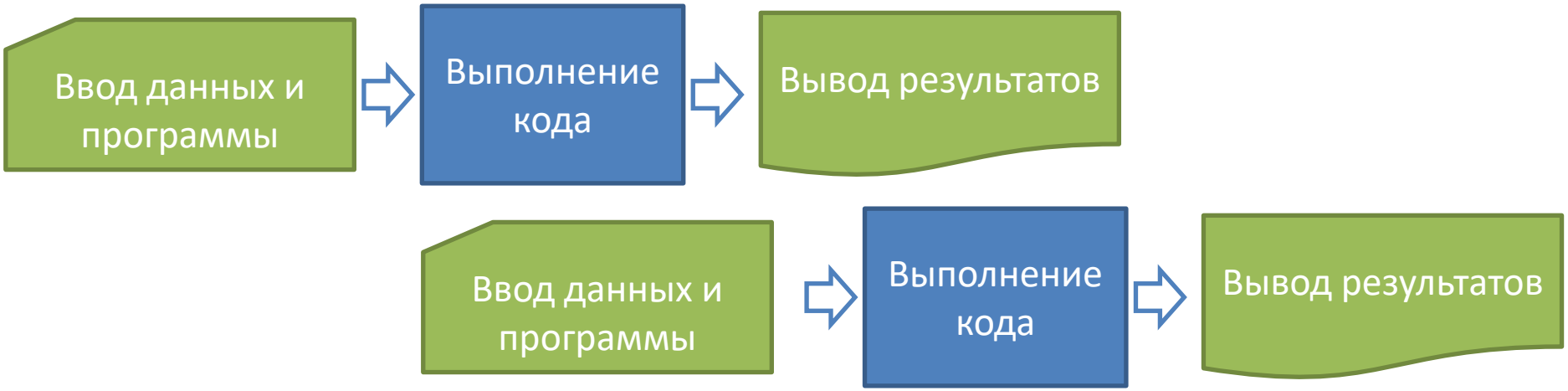
История

- 1960-е – Появление ОС, реализующих межпроцессное взаимодействие
- 1960-е – ARPANET и далее Usenet, FidoNet, Internet…
- 1960-1970-е – появление многопроцессорных систем
- 1970-е – Широкое распространение ЛВС на базе Ethernet
- 1980-е – Распределенные вычисления становятся активно изучаемой и обсуждаемой темой

История ОС – 1/3

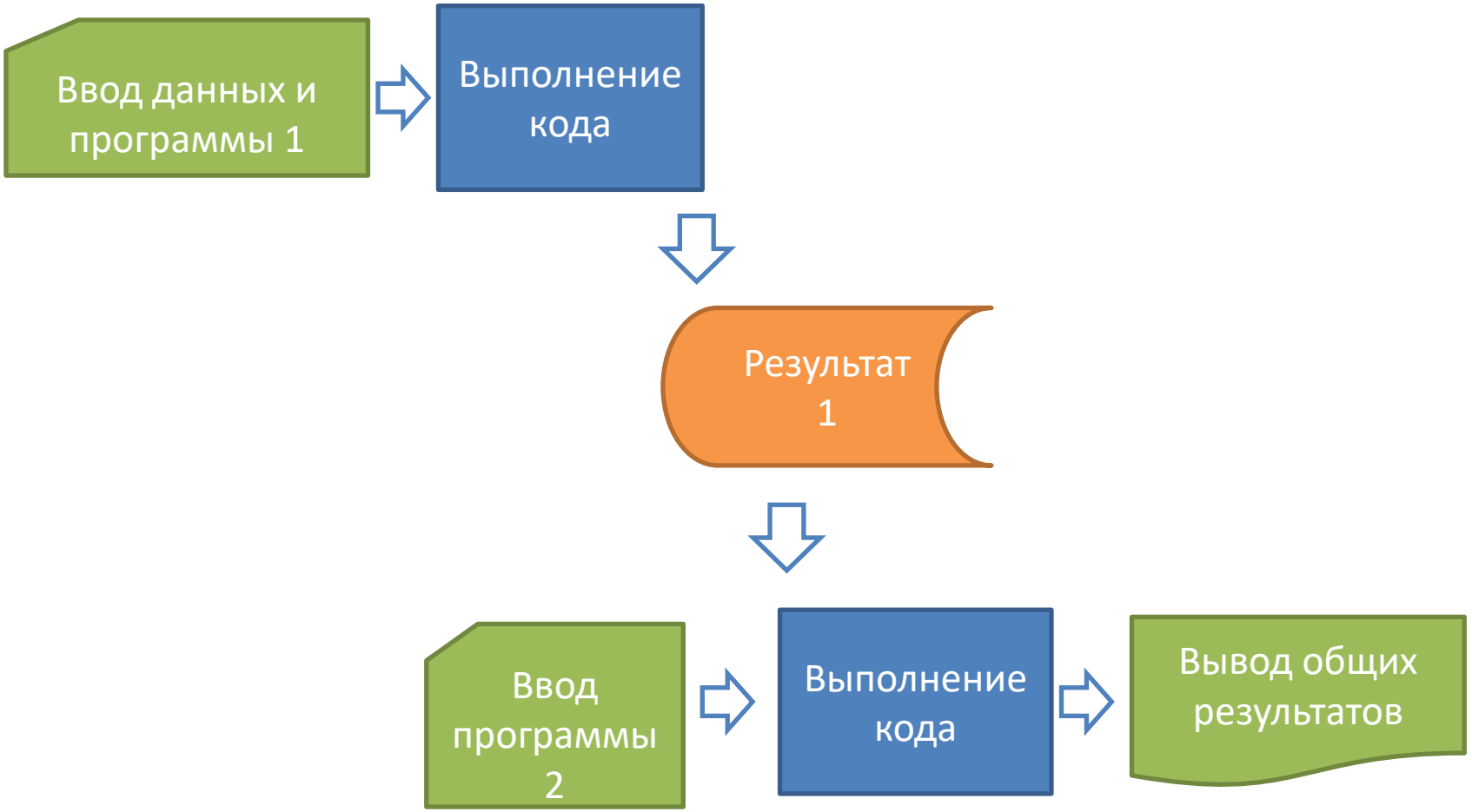
Последовательный запуск задач

Пакетный запуск задач



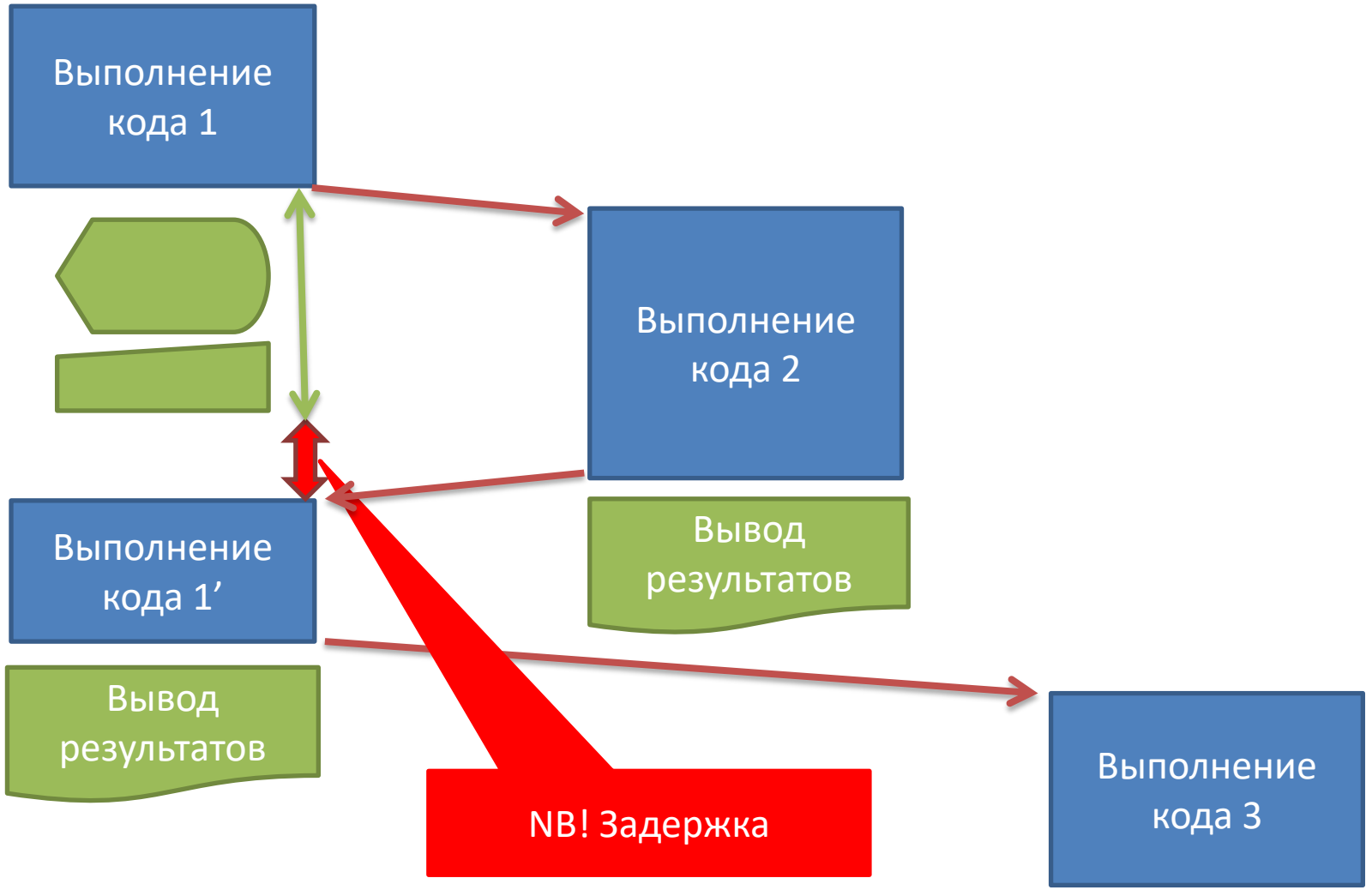
Зарождение ОС – 2/3

Последовательная обработка данных



Зарождение ОС – 3/3

Параллельное выполнение



Общее описание

Классификация

Виртуальная память процесса

ПРОЦЕССЫ

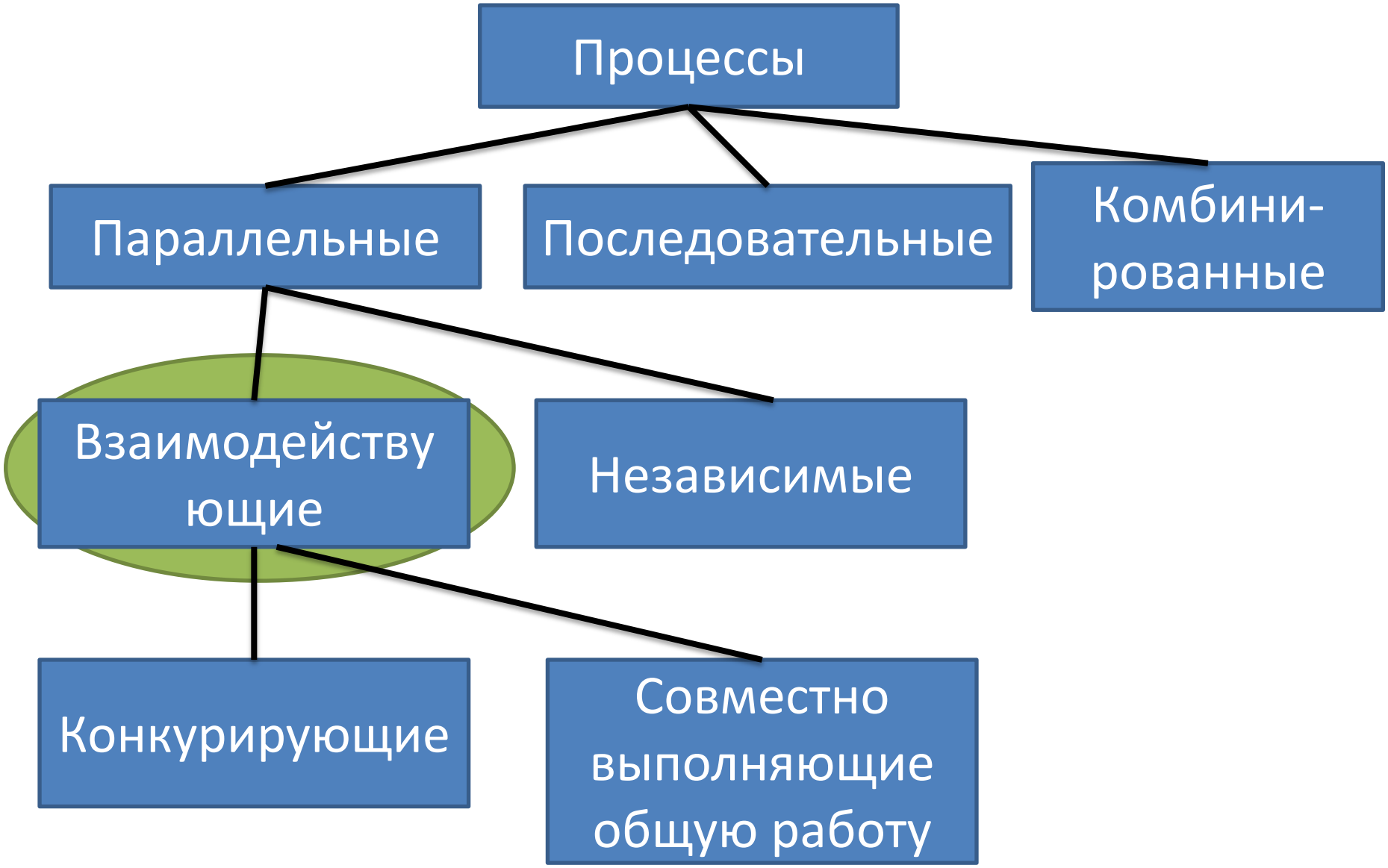
Процесс – 1/2

- Процесс – программа, которая выполняется в текущий момент
  - Находится в оперативной памяти (возможно, не полностью)
  - Все ПО, исполняемое на компьютере, а также часть ОС, организовано в виде процессов
  - Программа vs. Процесс:
    - Компьютерная программа – пассивная последовательность инструкций [хранящаяся на диске где-то]
    - Процесс — это непосредственное выполнение этих инструкций

Процесс – 2/2

- ISO 9000:2000 → Совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие

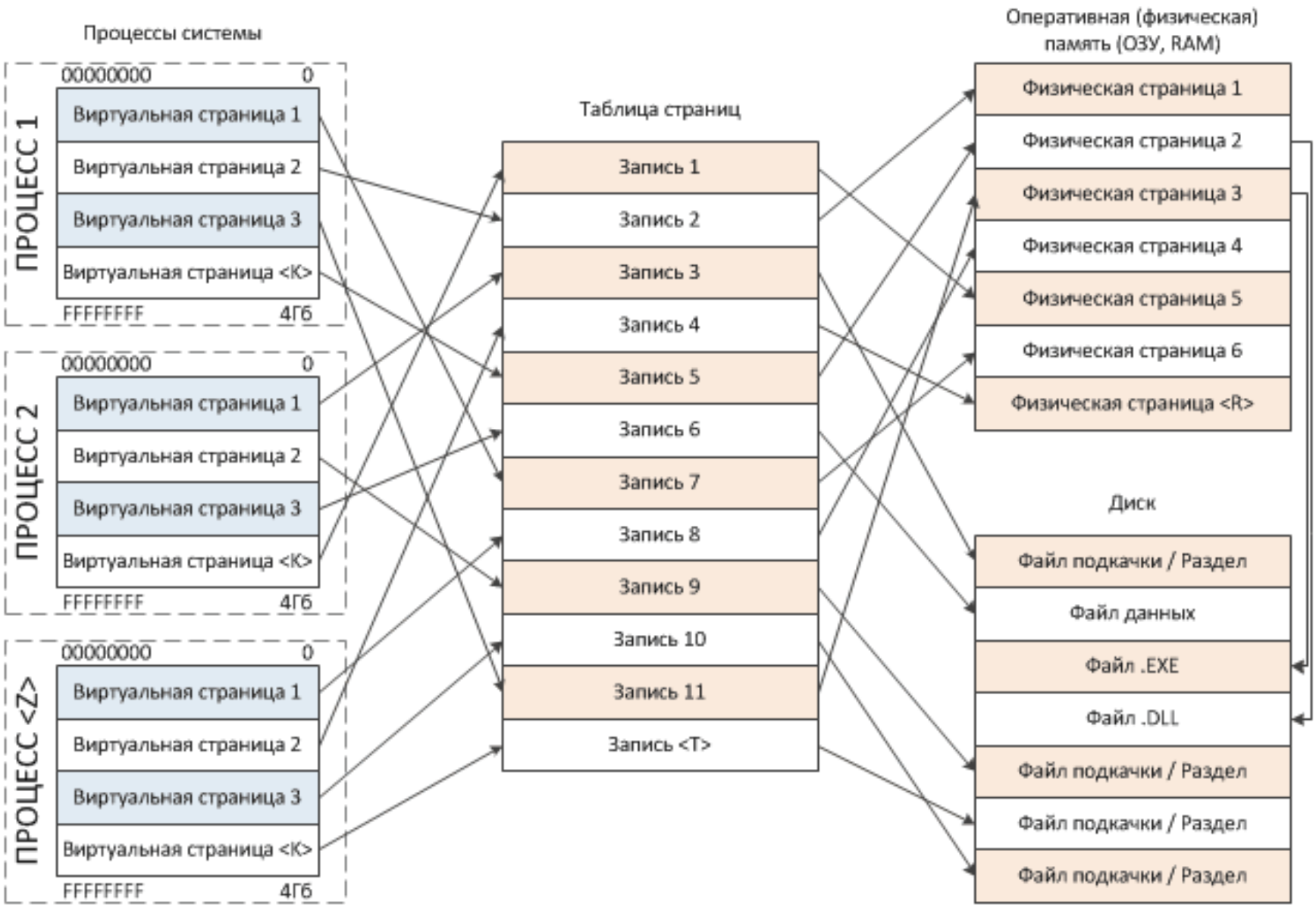
Классификация процессов



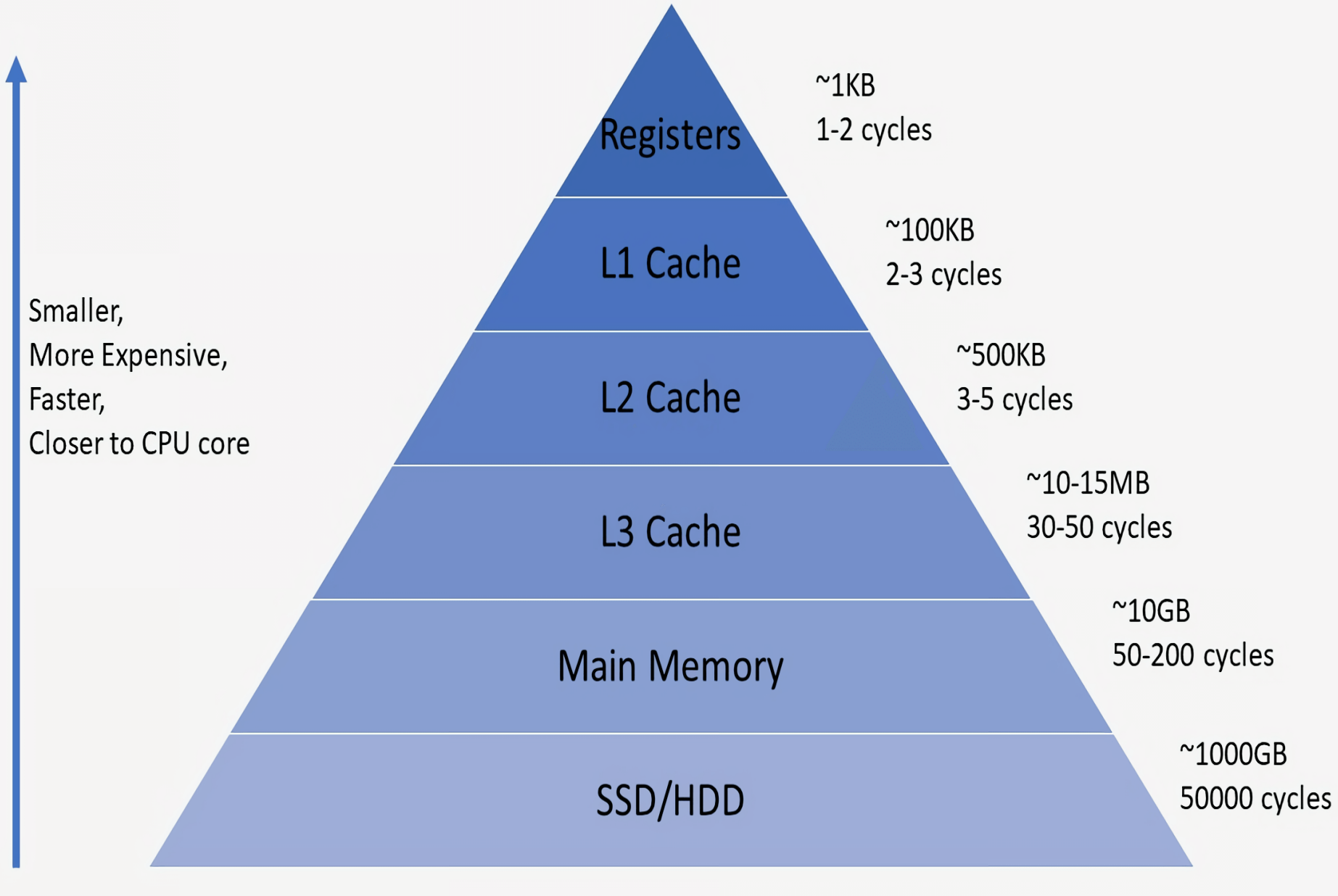
Что входит в процесс

- В процесс входят элементы выполняющейся программы:
  - Адресное пространство (принадлежность процесса, возможно совместное управление)
  - Глобальные переменные (управляются ОС)
  - Регистры [процессора]
  - Стек
  - Открытые файлы и т. д.

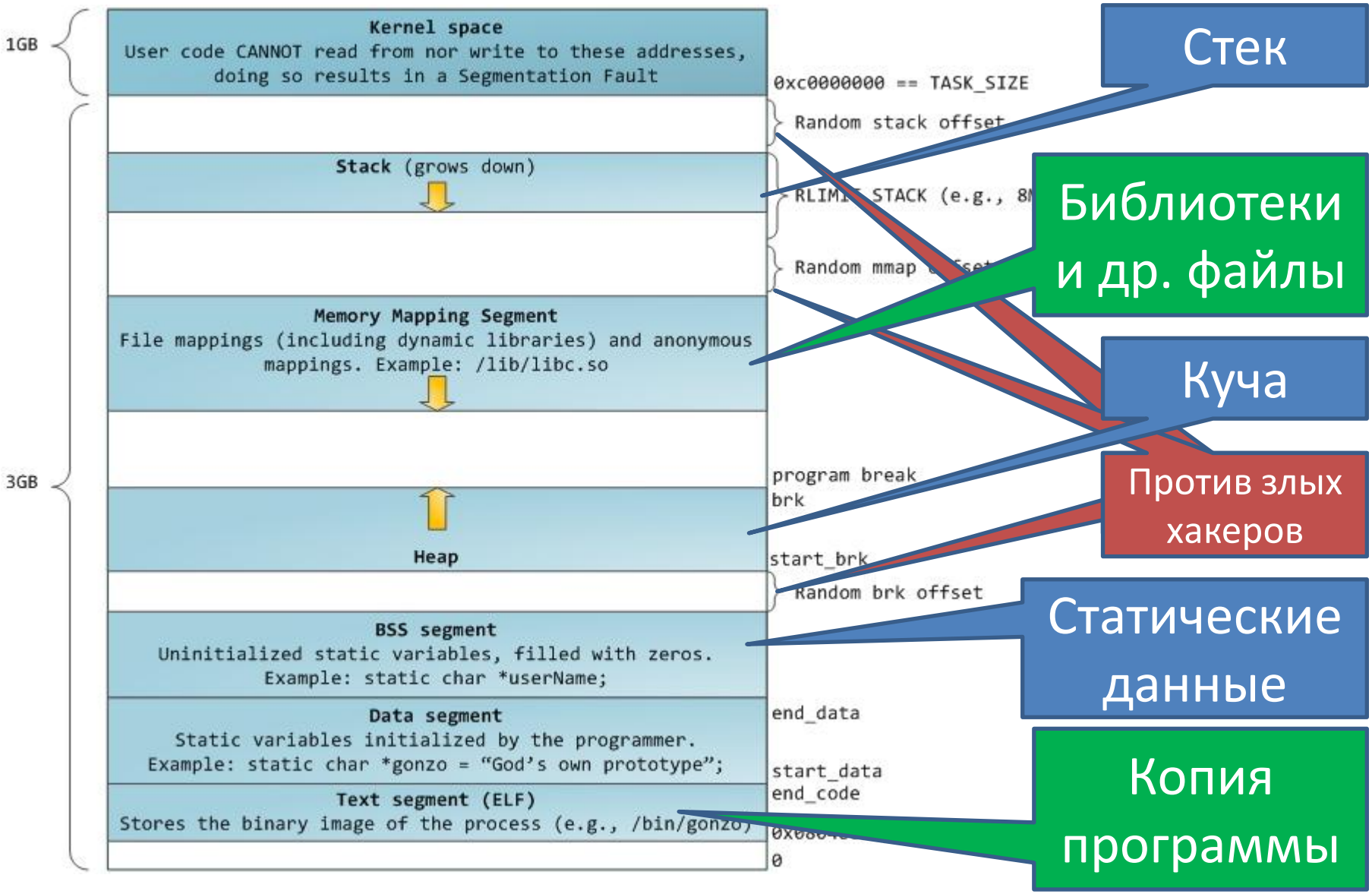
Виртуальная память: страничная организация



Скорость доступа к памяти



Память процесса



Память процесса: Стек

- Используется для хранения списка адресов возврата:
  - При вызове функции создается пакет данных:
    - Кто (откуда, с какого адреса вызвал)
    - С какими параметрами
  - При вызове добавляется новый (LIFO)
  - При возврате из функции уничтожается
- ЯП высокого уровня создают эти пакеты автоматически
- Стек создается для каждого потока

Память процесса: Куча

- Куча (heap) — структура данных, с помощью которой реализована динамически распределяемая память приложения
- Выделяется при создании процесса
- В процессе работы может быть увеличена

Что описывает процесс

- Описание процесса включает:
  - Имя
  - PID
  - Ссылку на индивидуальное адресное пространство, содержащее:
    - Исполнимый модуль
    - Данные, включая переменные
  - Данные ЦП, включая:
    - Счетчик команд (указатель на следующую команду для выполнения)
    - Содержимое регистров ЦП

Задачи управления

Иерархии процессов

УПРАВЛЕНИЕ ПРОЦЕССАМИ

Управление процессами

- При управлении процессами ОС выполняет следующие операции:
  - Создание процесса
  - Переключение контекстов, включая Планирование процессов
  - Удаление процесса

Иерархия процессов

- В некоторых системах родительский и дочерний процессы остаются связанными между собой определенным образом
  - Дочерний процесс также может, в свою очередь, создавать процессы, формируя иерархию процессов
- Кроме PID (process identifier) вводится PPID (parent process identifier)
- Сигналы ОС могут доставляться всем связанным процессам в иерархии

Иерархия процессов: UNIX

- В UNIX при старте ОС создается единый процесс-прародитель. Все остальные могут быть порождены только какими-либо другими процессами → Создается единое дерево иерархии
- Если процесс-родитель завершает свою работу до завершения выполнения процесса-ребенка, PPID для процесса-ребенка заменяется на идентификатор корневого процесса (процесса init)

Иерархия процессов: Windows

- Иерархии нет, но:
  - Для управления дочерними процессами используются хэндлы, получаемые при их запуске
  - Хендл может быть получен и другими процессами, не только родительским
    - Например, с помощью OpenProcess())
  - Есть группы процессов
    - Параметр функции CreateProcess() Create_New_Process_Group указывает на то, что создаваемый процесс является корневым для новой группы процессов, которые могут управляться группой

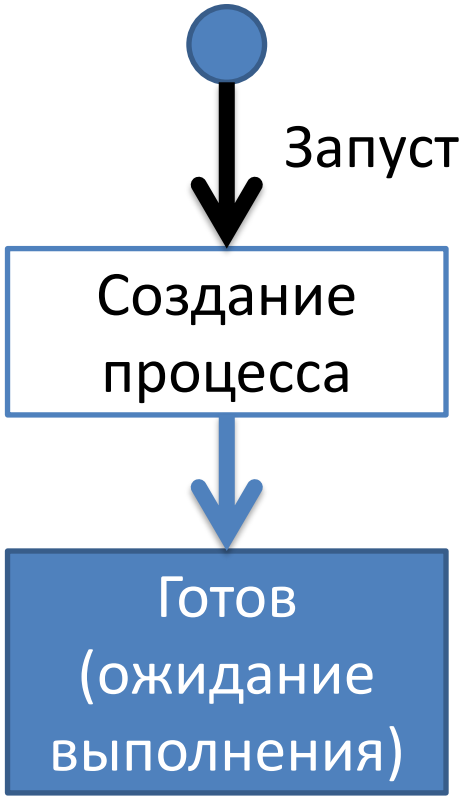
Когда

Кто

Что происходит

СОЗДАНИЕ ПРОЦЕССА

Состояния процесса – 1/4



Инициатива:

ОС

Другого процесса, пользователя, ОС

Создание процесса – когда

- При инициализации системы
- В произвольный момент уже работающий процесс выполняет системный запрос на создание [другого] процесса. Например:
  - Запрос пользователя на создание процесса
  - Инициализация пакетного задания

Создание процесса – как, кем

- Новый процесс формируется на основании системного запроса от текущего процесса
- Текущим процессом может быть:
  - Процесс, запущенный пользователем
  - Системный процесс
  - Процесс, инициализированный клавиатурой или мышью
  - Процесс, управляющий пакетами

Создание процесса – что (операции)

- Присвоить уникальный PID
- Выделить и инициализировать (частично заполнить нужной информацией) виртуальную память процесса
- Инициализировать PCB (process control block – блок управления процессом)
- Добавить процесс в очередь «готовых к выполнению»

Создание процесса в *NIX – 1/2

1. Обязательно: **`fork()`** (ветвление)
   - Создает дубликат вызываемого процесса
   - PCB дочернего процесса такой же, как PCB родителя, кроме: PID – уникальный, PPID = PID родителя
   - Адресное пространство копируется
     - Это долго и сложно. В Linux используется подход «copy-on-write»: все страницы помечаются как read-only, при попытке записи (родителем или клоном) страница копируется, флаг снимается, запись происходит в копию
2. Почти всегда: **`exec()`**
   - Заменяет образ процесса-клона новой программой, которая должна быть выполнена (переписывается виртуальная память процесса)

Создание процесса в *NIX – 2/2

- Преимущества:
  - Есть иерархия процессов
  - Копия «знает, что она копия» и может сразу выполнять отличные от родителя действия с теми же данными – они уже есть в виртуальной памяти
- Недостатки:
  - Операция `fork()` выполняется долго
  - Решение «copy-on-write» ускоряет выполнение `fork()`, но замедляет дальнейшую работу

Создание процесса в Windows

- Функция CreateProcess() интерфейса Win32
  - Управляет созданием процесса и запуском в нем нужной программы
  - Возвращает хендл созданного процесса

Истинная / псевдо- параллельность

Кто переключает

Что происходит

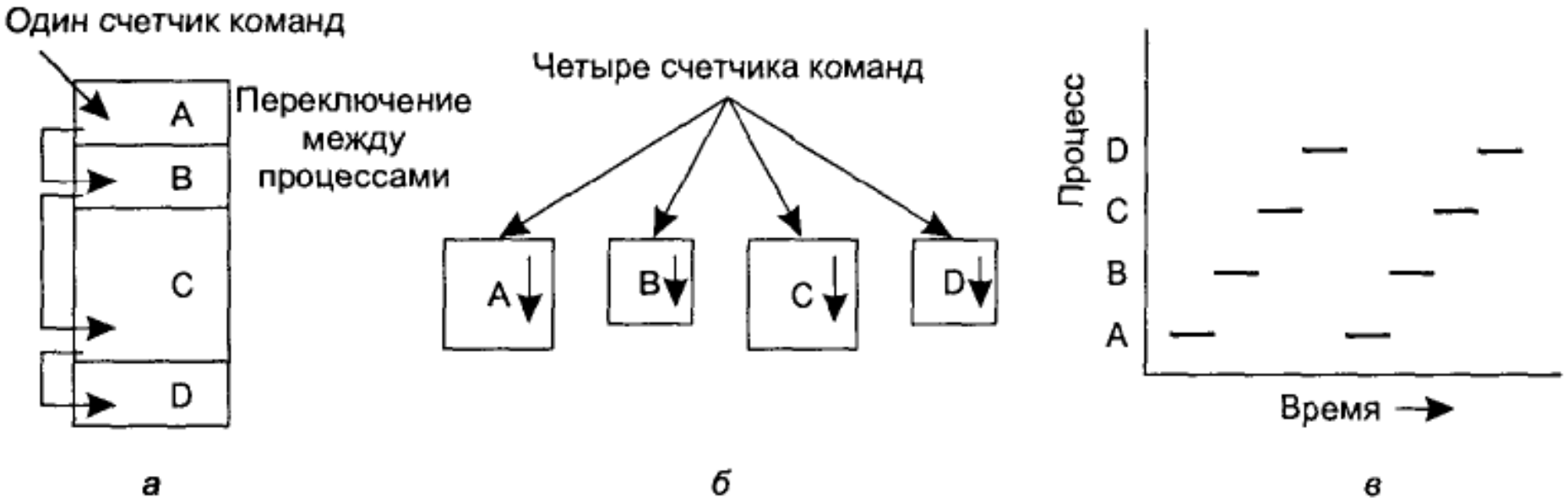
Планирование задач

ПЕРЕКЛЮЧЕНИЕ КОНТЕКСТОВ

Параллельность выполнения

- Параллельность процессов может быть реализована в виде:
  - Истинной параллельности выполнения:
    - На разных хостах
    - На одном хосте: выполнение процессов на разных ЦП (ядрах)
  - Псевдопараллельного выполнения процессов на одном хосте
    - ОС выделяет каждому процессу некоторое время ЦП (обычно, по несколько миллисекунд), после чего [насильно (?)] передает управление другому

Псевдопараллельность



Четыре программы, работающие в многозадачном режиме

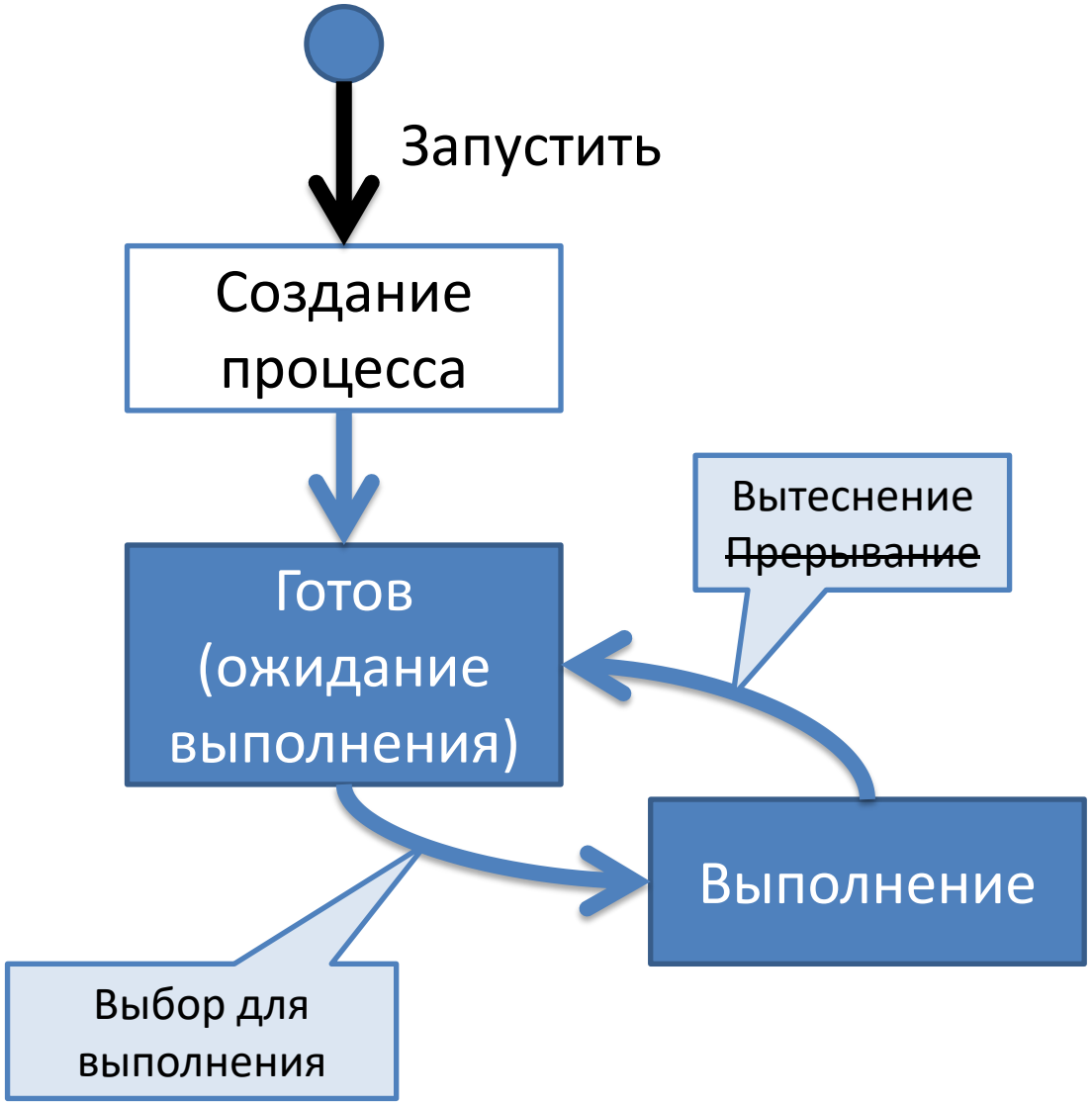
Концептуальная модель четырех независимых друг от друга последовательных процессов

В отдельно взятый момент времени активна только одна программа

Многозадачность vs параллельность

- В современных [настольных] ОС многозадачность реализована путем псевдопараллельности →
  - Обычно, число процессов много больше числа доступных ядер (см. выше)
  - Истинная параллельность также присутствует в виде выполнения процессов на разных ЦП (ядрах)
  - ОС суперкомпьютеров строятся так, чтобы число процессов соответствовало числу ЦП, чтобы снизить непроизводительные расходы времени ЦП на переключение контекстов

Состояния процесса – 2/4



Инициатива:

ОС

Другого процесса, пользователя, ОС

Переключение контекста – 1/4

- Переключение контекста включает:
  - Вытеснить (прервать выполнение):
    - Выгрузить (сохранить) содержимое регистров ЦП
    - Очистить конвейеры команд и данных ЦП
    - Очистить TLB (Translation lookaside buffer – Буфер ассоциативной трансляции)
  - Выбрать на выполнение:
    - Загрузить предварительно сохраненное содержимое регистров ЦП

Врезка: TLB – 1/2

- TLB – кэш ЦП, используемый для ускорения перевода адреса виртуальной памяти в адрес физической памяти
  - используется всеми современными процессорами, поддерживающих страничную организации памяти
  - содержит фиксированный набор записей (от 8 до 4096) и является ***ассоциативной памятью*** (адресуемой по содержимому)

Врезка: TLB – 2/2

- Запись TLB содержит соответствие адреса страницы виртуальной памяти адресу физической памяти. Если адреса в TLB нет, то **ЦП** (sic!):
  - ищет по таблицам страниц
  - сохраняет полученный адрес в TLB
- При этом:
  - поиск занимает в 10…60 раз больше времени, чем чтение из TLB
  - Вероятность промаха ~ 0,01%...1 %

Переключение контекста – 2/4

- После переключения:
  - Содержимое кэша ЦП оказывается [совершенно] неприменимым к новому процессу:
    - Кэш данных
    - Кэш команд

    → Пока не заполнятся, работает чуть медленнее

    → Если попытаться не очищать, то процесс может получить доступ к данным другого процесса, что нарушает всю идею
  - Если происходит переключение контекста на процесс, который до этого долгое время не использовался, то в физической памяти может остаться мало страниц из виртуальной памяти процесса

    → Работает сильно медленнее, так как прерывает выполнение процесса на подгрузку страниц в ОЗУ

Переключение контекста – 3/4

- Кто [прерывает выполнение]?
  - Добровольное (voluntary): выполняющийся процесс/поток сам передает управление
    - Проблема: может и не отдать ^_^
    - Кооперативная многозадачность
    - Примеры: Windows 1.0 – Windows 95/Vista
  - Принудительное (non-voluntary): ядро ОС насильно отбирает управление
    - Проблема: закон подлости!
    - Вытесняющая многозадачность
    - Все современные ОС, Windows начиная с NT и XP

Планирование выполнения задач – 1/4

- Планирование заключается в назначении приоритетов процессам в очереди с приоритетами. Эту задачу выполняет планировщиком (scheduler)
  - NB! Планировщик часть ядра ОС, НЕ оформленная в виде процесса
  - Цель планирования задач: обеспечение наиболее полной загрузки процессора

Планирование выполнения задач – 2/4

- Показатели:
  - Производительность — количество процессов, которые завершают выполнение за единицу времени (sic!)
  - Время ожидания — время, которое процесс ожидает в очереди готовности
  - Время отклика — время, которое проходит от поступления первого запроса в потоке запросов до получения ответа на первый запрос

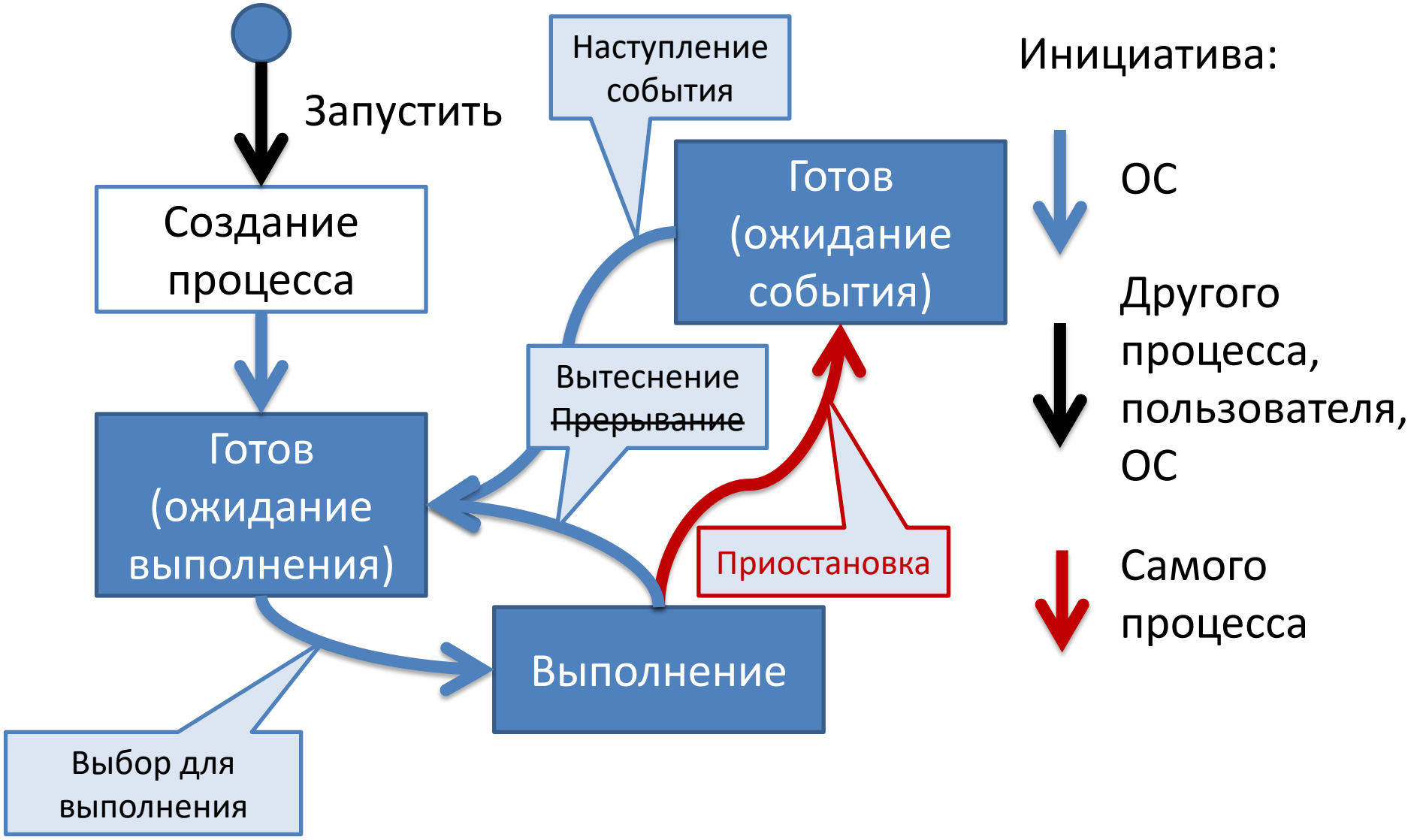
Планирование выполнения задач – 3/4

- Виды планировщиков:
  - Долговременный планировщик
    - Решает, какие процессы будут добавлены в очередь готовых процессов
    - Ограничивает [возможную] загрузку системы
  - Среднесрочный планировщик
    - Управляет подкачкой: выгружает процесс, если он:
      - был неактивен некоторое время;
      - имеет низкий приоритет;
      - часто вызывает ошибки страниц (page fault);
      - занимает большое количество основной памяти

Планирование выполнения задач – 4/4

- Виды планировщиков (продолжение):
  - Краткосрочный планировщик ~ Диспетчер
    - Выбирает процесс для запуска на ЦПУ
    - Занимается непосредственно переключением контекста
    - NB! Код диспетчера должен быть очень быстродействующим, так как он осуществляет каждое переключение процессов

Состояния процесса – 3/4



Переключение контекста – 4/4

- Кто и почему прерывает выполнение?
  - ОС [нарочно]: Истек квант времени, выделенный процессу на выполнение
  - Процесс:
    - Неявно: Выполнение блокирующего системного вызова
      - Пример: запрос на ввод-вывод
    - Явно: Выполнение системного вызова, приостанавливающего выполнение процесса до наступления события (select, poll, epoll, pause, wait,…) либо момента времени (sleep, nanosleep,..)
    - Нарочно: Процесс использует синхронизирующие примитивы ядра

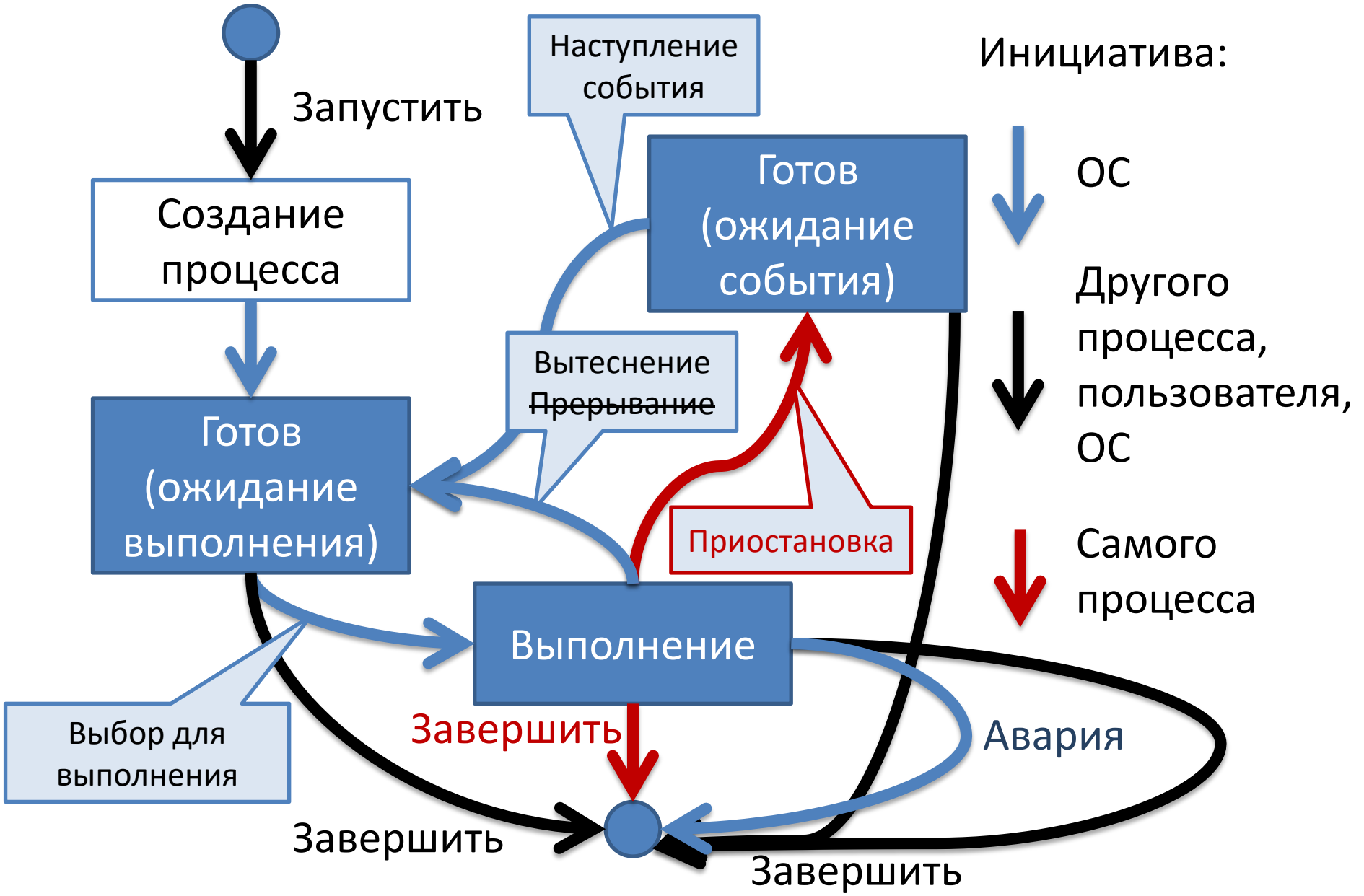
Причины

Кто

Что происходит

ЗАВЕРШЕНИЕ ПРОЦЕССА

Состояния процесса – 3/3



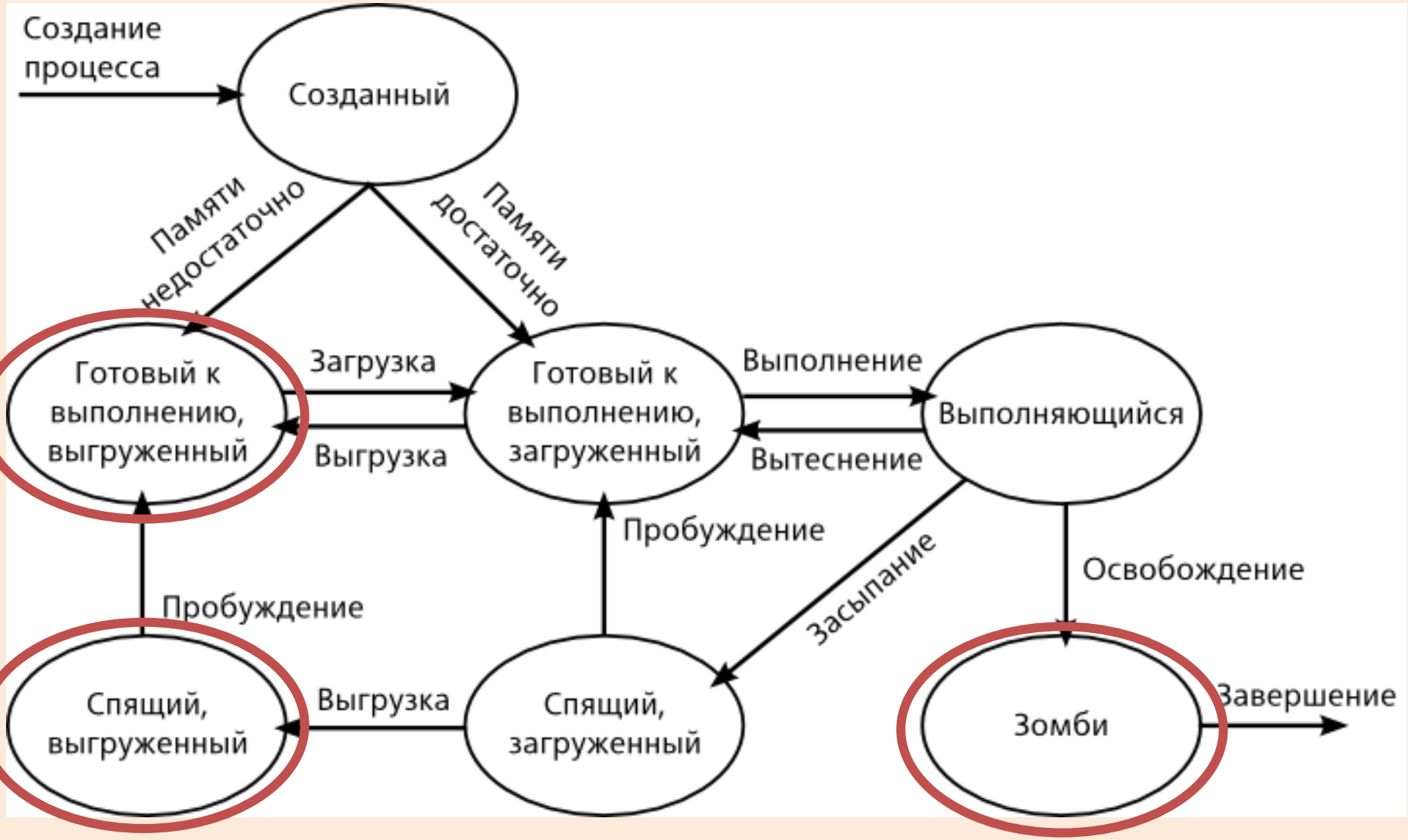
Завершение процесса – 1/2

- Преднамеренное:
  - «Обычный» выход по инициативе самого процесса
  - Выход по ошибке по инициативе самого процесса
  - Уничтожение другим процессом
- Непреднамеренное:
  - Выход по неисправимой ошибке
  - Уничтожение другим процессом

Завершение процесса – подробности

- UNIX: системный запрос exit()
- Windows: функция ExitProcess()

Бонус: Состояния процесса UNIX – 1/2



Бонус: Состояния процесса UNIX – 2/2

- Большинство состояний процесса совпадает с классическим набором состояний процессов
  - Внимание: выгруженный/загруженный
- Для ОС UNIX характерно особое состояние процесса – **зомби**:
  - Процесс становится зомби если он завершился раньше, чем этого ожидал его родительский процесс
  - Отслеживание зомби помогает корректно завершать группы процессов, освобождая ресурсы

Назначение

Управление

Устройство

ПОТОКИ

Потоки (threads)

- Поток [выполнения] (нить, thread) – наименьшая единица обработки, исполнение которой может быть назначено ядром ОС
  - Реализация потоков в разных ОС отличается, но в большинстве случаев поток находится внутри процесса
  - Несколько потоков могут существовать одновременно в рамках одного процесса и совместно использовать его ресурсы:
    - Память
    - Инструкции процесса (его код)
    - Контекст (значения переменных, которые они имеют в любой момент времени)

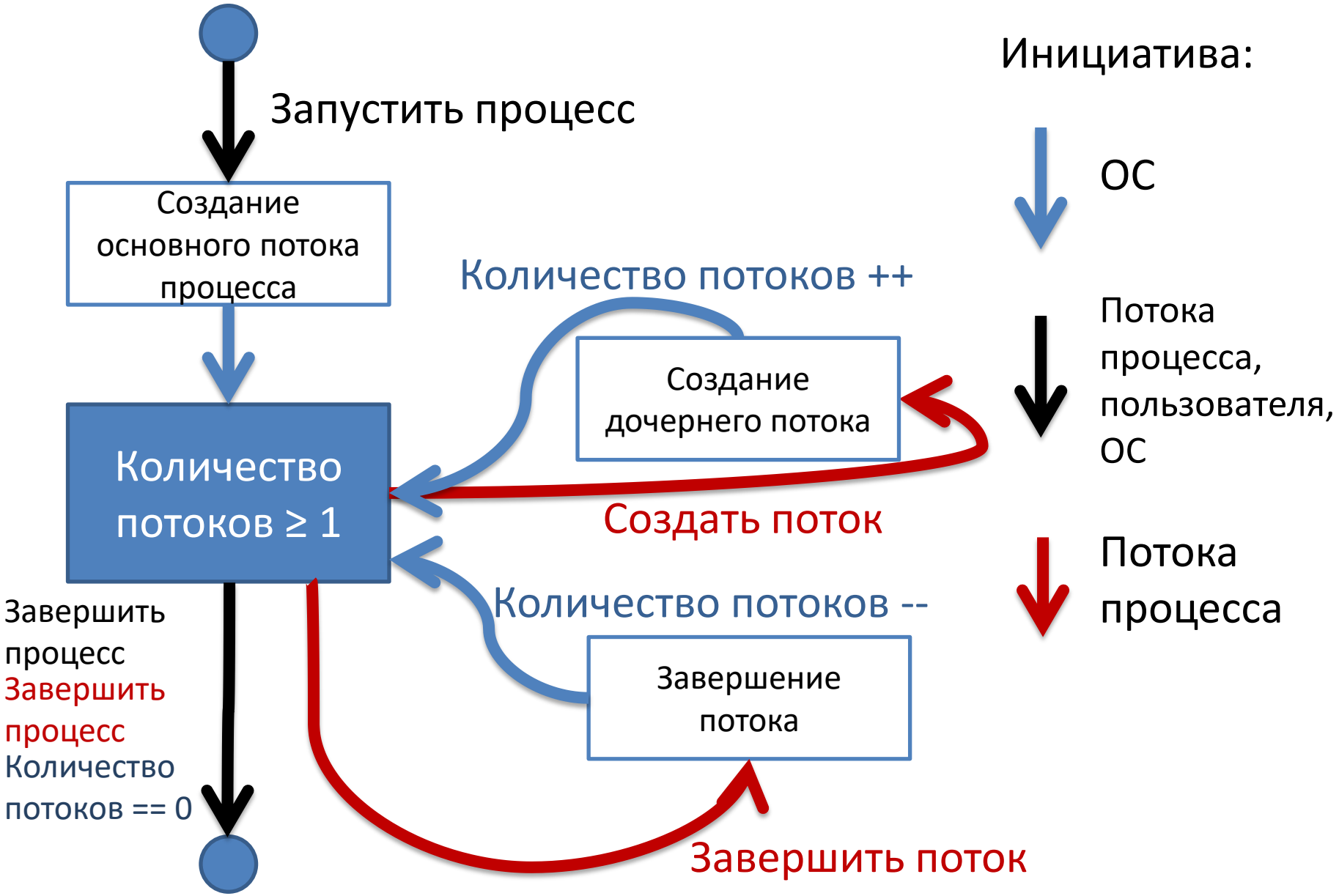
Выполнение потоков

- Многозадачные ОС также обеспечивают и многопоточность:
  - На одном процессоре: путём переключения между потоками процесса
  - В многопроцессорных и многоядерных системах: потоки выполняться истинно параллельно, при этом каждый процессор или ядро обрабатывает отдельный поток

Управление потоками – 1/4

- Основными задачами управления потоками являются:
  - Создание
  - Переключение
  - Уничтожение
- Создание:
  - Основной поток процесса создается автоматически ОС при создании процесса
  - Дочерние потоки создаются по команде основного потока или других потоков
    - Дочерние потоки могут создаваться активными и неактивными (ожидающими события / активации)

Создание/удаление потоков



Завершение процесса vs. потока

- Как связаны потоки и процесс?
  - Завершение первичного потока процесса завершает процесс
    - Однако не всегда – например, в Windows первичный поток может отменить завершение работы других потоков, при помощи явного вызова **ExitThread()** перед своим завершением
  - Любой поток процесса может в явном виде потребовать завершения всего процесса
  - Существуют ОС, в которых возможна передача потока между процессами (в том числе выполняющимися на разных хостах!)

Завершение процесса Windows – 1/3

- Процесс выполняется, пока:
  - Любой поток процесса вызывает функцию **ExitProcess()** – она [аккуратно] завершает работу всех потоков процесса
  - Первичный поток процесса возвращает значение
    - Если первичный поток может отменил завершение работы других потоков, вызвав **ExitThread()** перед возвратом своего значения, то один из оставшихся потоков может все еще вызвать **ExitProcess()**, чтобы явно завершит остальные потоки

Завершение процесса Windows – 2/3

- Процесс выполняется, пока (2):
  - Последний поток процесса не завершил работу
  - Поток процесса вызывает функцию **TerminateProcess()**, с дескриптором процесса – это завершает работу всех потоков процесса, без разрешения выполнения завершающих действий (например, очистки памяти или сохранения данных)

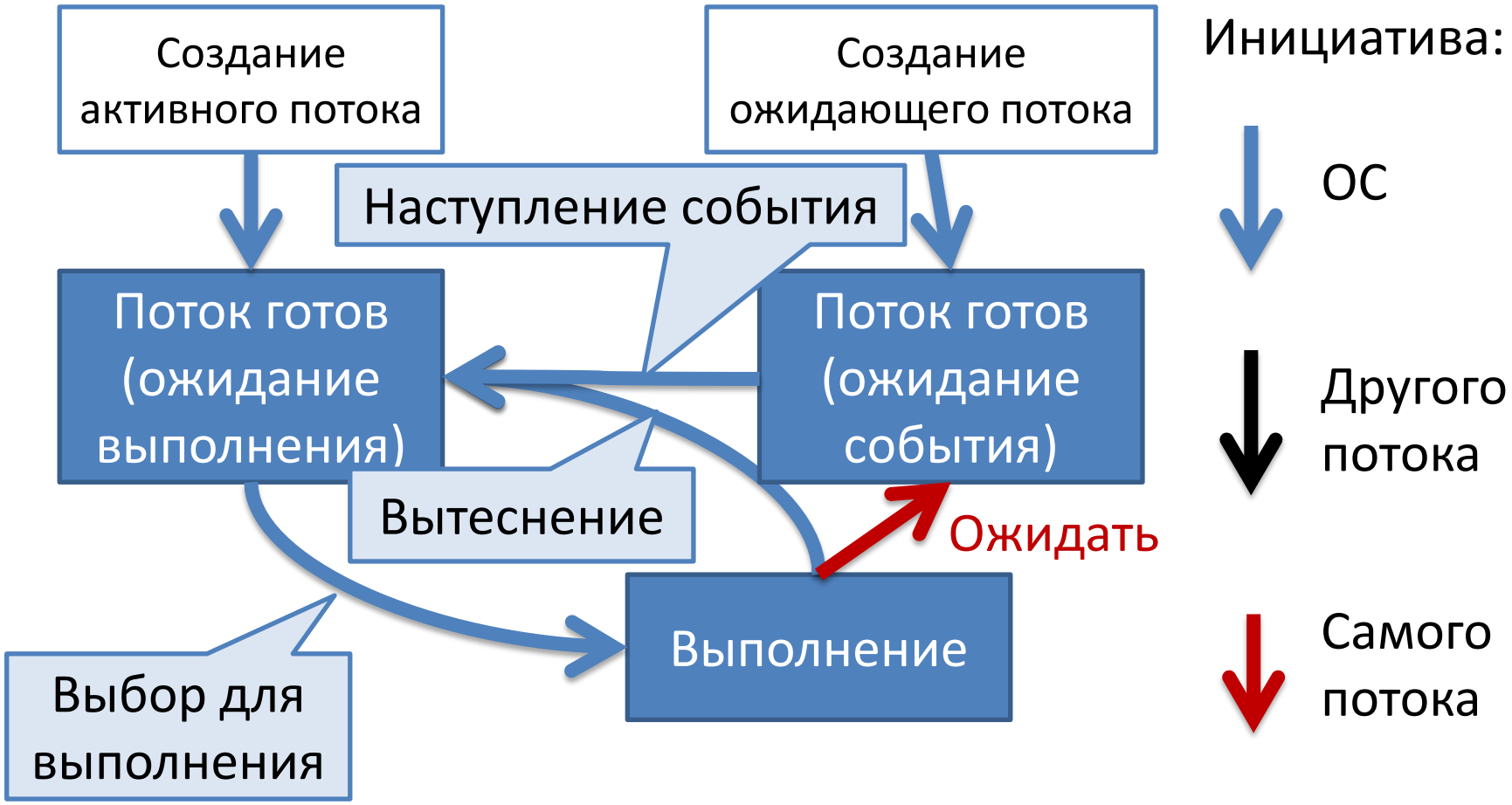
Завершение процесса Windows – 3/3

- Процесс выполняется, пока (3):
  - У консольных процессов, есть заданная по умолчанию функция обработчика сигналов
    - По умолчанию, она вызывает **ExitProcess()**, когда нажаты **[CTRL+BREAK]** или **[CTRL+C]**
    - Все консольные процессы, связанные с этой консолью получают эти сигналы
    - Отсоединенные процессы и процессы с GUI не реагируют
  - ОС выключается или пользователь заканчивает работу сессии

Управление потоками – 2/4

- Переключение:
  - В-основном осуществляется ядром ОС аналогично переключению контекстов процессов
  - Поток может инициировать приостановку своего выполнения:
    - Приостановиться до наступления внешнего события
    - Приостановиться до наступления момента времени
  - Поток может активировать другой поток (прервать его ожидание)

Переключение потоков



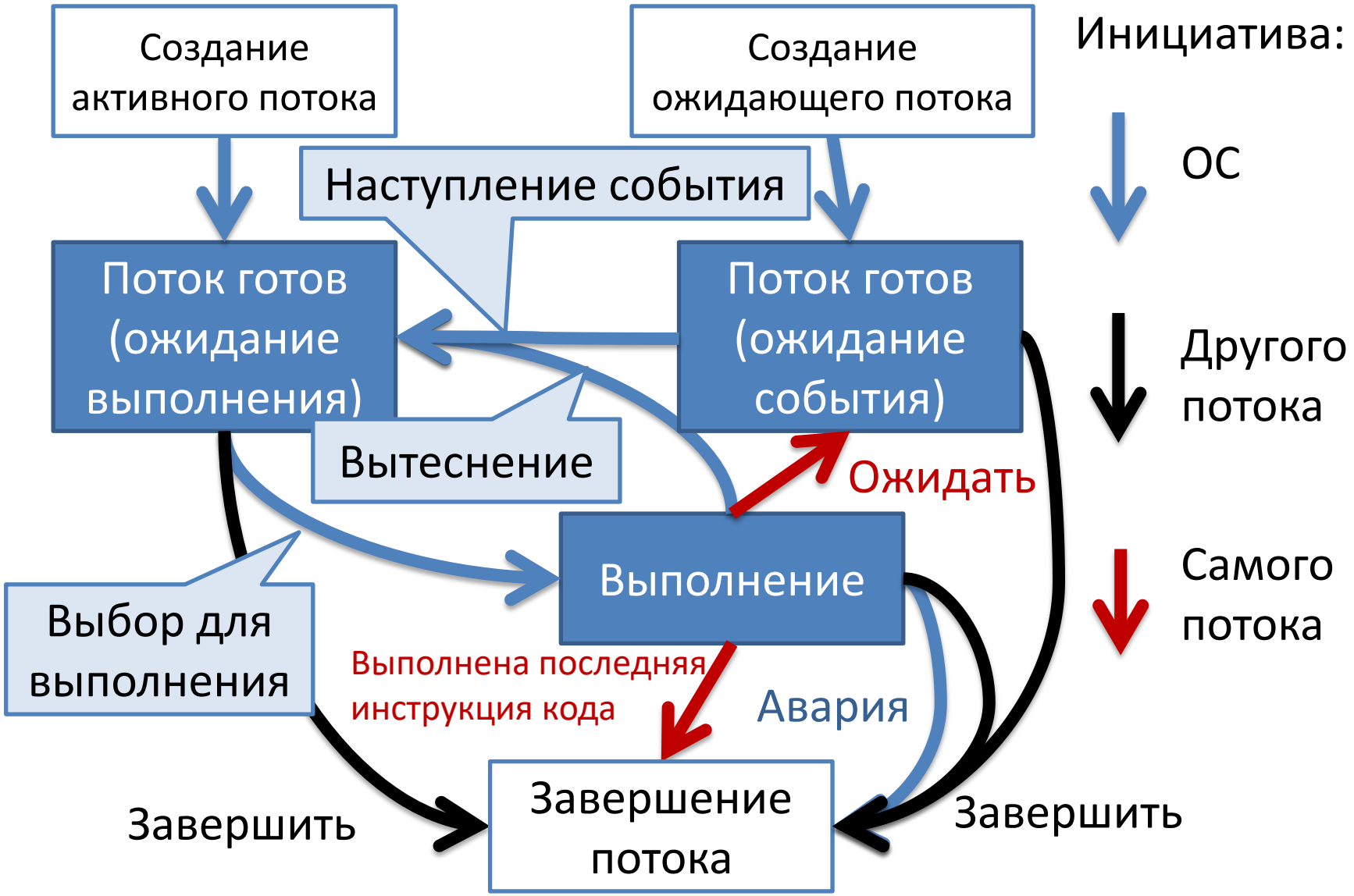
Управление потоками – 3/4

- Завершение потока (1/2):
  - Поток завершает выполнение своего кода (тела функции потока)
  - При выполнении кода потока происходит авария (NB!)
  - Поток, другой поток, ОС выдает команду на остановку процесса и ОС прерывает все потоки

Управление потоками – 4/4

- Завершение потока (2/2):
  - Другой поток выдает сигнал на остановку. При этом:
    - Поток должен быть останавливаемым (в POSIX: cancel state – enabled)
    - Поток может прерваться:
      - в любой момент (в POSIX: cancel type – deferred)
      - должен вызвать команду, трактуемую как возможная точка остановки (cancellation point), то есть приостановиться (в POSIX: cancel type – asynchronous)

Остановка потоков



ПРОЦЕССЫ, ПОТОКИ, ФАЙБЕРЫ

Эволюция процессов – 1/3

- Постановка задачи:
  - Необходимо выполнять несколько задач одновременно
  - Данные одних задач должны быть защищены от других задач
- Решение: Процессы!

Эволюция процессов – 2/3

- Постановка задачи:
  - Процессы слишком сложны, чтобы выполняться в виде единого алгоритма (потока управления)
  - Обмен данными между процессами сложен
- Решение: Потоки (Threads)

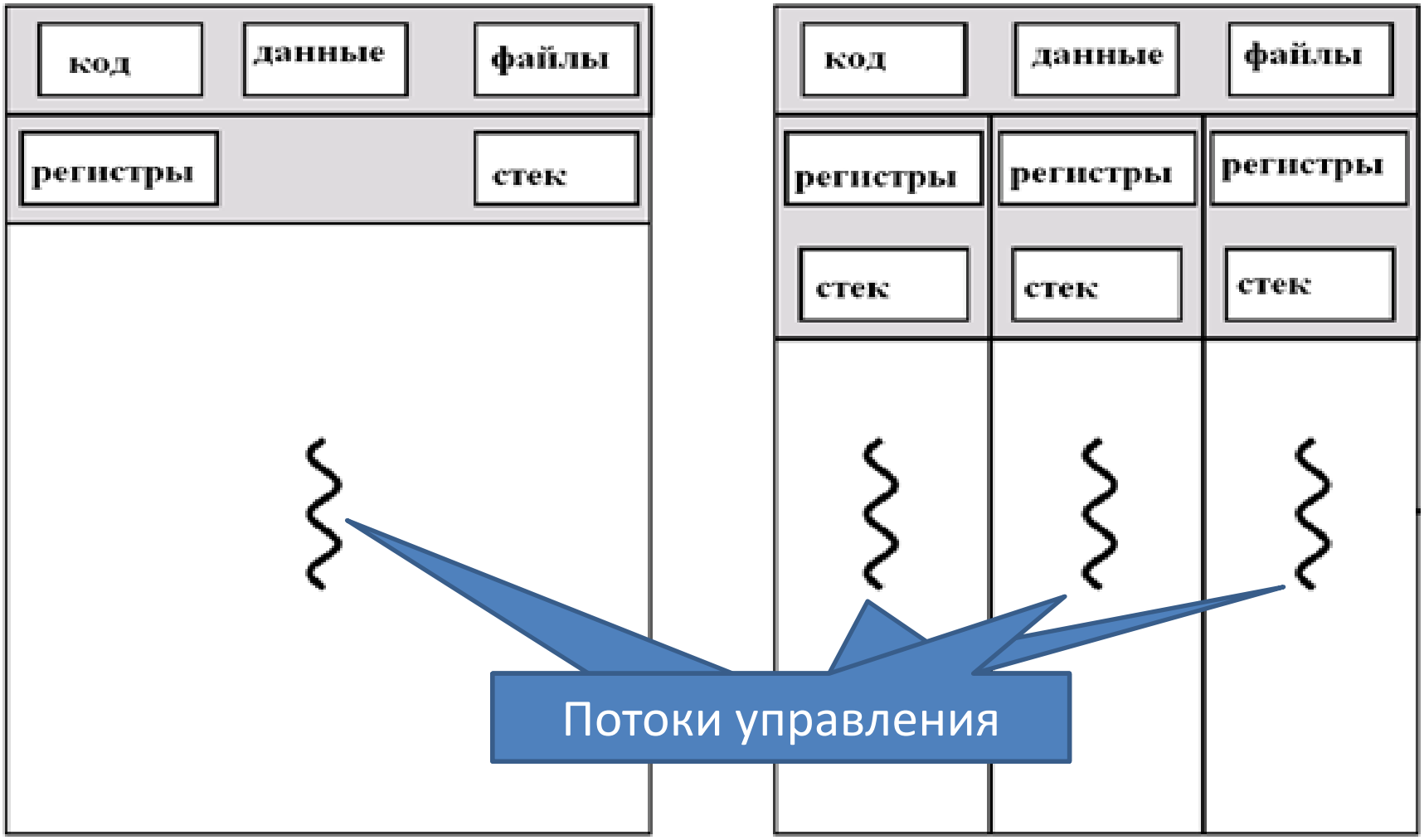
Эволюция процессов – 3/3

- Постановка задачи:
  - Потоки управляются ОС способом, неэффективном в каждом конкретном случае
- Решение: Файберы (Fibers)

Сравнение

	Process	Thread	Fiber
Кто управляет	ОС	ОС	Приложение (Среда параллельного программирования)
Обмен данными	IPC	Общая память процесса	Общая память процесса
Многозадачность	Вытесняющая	Вытесняющая	Кооперативная
Достоинства	Есть везде	Есть везде Обмен данными «Легкие»	Эффективное планирование
Недостатки	«Тяжелые» Сложность обмена данными	Синхронизация доступа к данным	Сложность организации на многопроцессорных системах

Процессы и потоки



Однопоточный процесс **Многопоточный процесс**

POSIX

POSIX

- POSIX (portable operating system interface — переносимый интерфейс операционных систем) набор стандартов, описывающих:
  - Системный API ОС – интерфейсы между операционной системой и прикладной программой
  - Библиотеку языка С
  - Набор приложений и их интерфейсов
- Является международным государственным стандартом ISO/IEC/IEEE 9945

Назначение POSIX

- Обеспечение совместимости различных UNIX-подобных ОС → Стал основой стандарта Single UNIX Specification → соответствие Single UNIX Specification позволяет ОС использовать торговую марку UNIX
  - Примечание: может быть использован и для не-Unix систем
- Обеспечение переносимости прикладных программ на уровне исходного кода

Стандарт

- По состоянию на 2017 год текущим является стандарт POSIX.1-2008
  - IEEE Std 1003.1-2008, 2016 Edition
  - Свободный доступ: http://pubs.opengroup.org/onlinepubs/9699919799/

Состав стандарта

- Основные понятия и определения (the Base Definitions, Issue 7)
- Системные интерфейсы и заголовочные файлы (the System Interfaces and Headers, Issue 7)
- Оболочка и утилиты (the Commands and Utilities, Issue 7) — описание утилит и командной оболочки `sh`, стандарты регулярных выражений
- Обоснование (Rationale):
  - Необходимость стандарта
  - Основные принципы
  - Причины включения (или отказа от включения) возможностей в стандарт

POSIX-совместимые ОС – 1/3

- Сертифицированы:
  - AIX
  - HP-UX
  - IRIX
  - OS X/macOS (начиная с 10.5 Leopard)
  - Solaris
  - Tru64
  - UnixWare
  - QNX Neutrino
  - Inspur K-UX
  - Integrity

POSIX-совместимые ОС – 2/3

- По большей части совместимы:
  - Android
  - BeOS
  - Contiki
  - FreeBSD
  - Linux (большая часть дистрибутивов)
  - NetBSD
  - Nucleus RTOS
  - OpenBSD
  - OpenSolaris
  - VxWorks
  - … И многие другие

POSIX-совместимые ОС – 3/3

- Windows ?:
  - Cygwin – POSIX-совместимая среда разработки и выполнения для Windows
  - Windows Subsystem for Linux – прослойка для запуска Linux приложений на Windows 10:
    - Использует образ Ubuntu
    - Бета вышла в апреле 2016

POSIX.1c Потоки

- Стандарт IEEE Std 1003.1c-1995. Содержит:
  - Описывает типы, функции и константы языка C (pthread.h + библиотека thread). Входит около 100 процедур (с префиксом pthread_), объединенных в:
    - Управление потоками: создание, присоединение и т.д.
    - Мьютексы (mutexes)
    - Условные переменные (condition variables)
    - Синхронизацию потоков с использованием блокирования (locks) записи/чтения и барьеров (barriers)
- Примечание: семафоры POSIX могут работать с потоками POSIX, но описаны в другом стандарте (POSIX.1b, Real-time extensions (IEEE Std 1003.1b-1993) → префиксы процедур семафоров «sem_», а не «pthread_»

Обмен данными между процессами

Управление выполнением процессов

СПОСОБЫ ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ (IPC)

IPC

- Межпроцессное взаимодействие (inter-process communication, IPC) – обмен данными между потоками одного или разных процессов
  - Реализуется:
    - Механизмами ядра ОС
    - Процессом (процессами) использующим механизмы ОС и реализующим новые возможности IPC – middleware
  - Связывает процессы, выполняющиеся:
    - На одном хосте (ОС, middleware)
    - На нескольких хостах сети (middleware)

Способы взаимодействия – 1/6

Метод	Описание
Файл	Данные, хранящиеся на диске или созданные по запросу файловым сервером. Доступны нескольким процессам одновременно
Сигнал	Системное сообщение, посылаемое одним процессом другому (внутри хоста!): • Есть выбор между несколькими заранее определенными сигналами (POSIX – 28 шт.) • Не содержит данных • Используется для информирования о событиях (заранее согласованных) или управления выполнением • Б***о***льшая часть сигналов зарезервирована для управления

Способы взаимодействия – 2/6

Метод	Описание
Сокет	Данные пересылаются через сетевой интерфейс другому процессу • Нет разницы между локальным и распределенным обменом • Чаще реализуется обмен потоком байтов (stream-oriented) – TCP. Тогда протоколы более высокого уровня формируют пакеты • Есть сообщение-ориентированные протоколы (message-oriented) – UDP, SCTP

Способы взаимодействия – 3/6

Метод	Описание
Сокет домена Unix, IPC-сокет (Unix domain socket) ***POSIX ОС, Windows 10***	Похоже на обычный сокет, но обмен идет через ядро ОС • Поддерживают надёжную потоковую передачу и передачу датаграмм: упорядоченную и надёжную или неупорядоченную и ненадёжную • Используют файловую систему как адреса. Это позволяет двум различным процессам открывать один и тот же сокет для взаимодействия между собой. Однако, конкретное взаимодействие, обмен данными, не использует файловую систему, а только буферы памяти ядра

Способы взаимодействия – 4/6

Метод	Описание
Очередь сообщений (Message queue)	Образуется очередь сообщений (не байтов) • Реализуется ОС • Одновременно несколько процессов могут читать и писать в очередь, не взаимодействуя напрямую
Канал (Pipe)	Однонаправленный канал данных • Данные записанные в канал буферизируются ОС до чтения • Процесс имеет три канала: stdin(IN), stdout(OUT) и stderr(OUT) • Соединяет строго два процесса • Двусторонний обмен возможен при встречном использовании каналов

Способы взаимодействия – 5/6

Метод	Описание
Именованный канал (Named pipe) ***POSIX ОС, Windows***	Канал, реализованный как файл • Может быть создано любое количество • Одновременно может использоваться 0..* процессами • Удаляется как обычный файл
Разделяемая память (Shared memory) ***POSIX ОС, Windows***	ОС помечает часть физической памяти как именованную область. Процессы могут отобразить ее в свое виртуальное адресное пространство • В Linux выглядит как обычный файл (может остаться при перезагрузке компьютера) • В Windows является артефактом ОС

Способы взаимодействия – 6/6

Метод	Описание
Обмен сообщениями (Message passing)	Подход, позволяющий [с помощью middleware] передать сообщение процессу. Предполагается, что вызов «правильной» процедуры обработки сообщения остается на совести адресуемого процесса и/или middleware Парадигмы: RPC, RMI и MPI Реализации: Java RMI, CORBA, DDS, MSMQ, MailSlots, QNX, etc.
Проецируемый в память файл (Memory-mapped file) ***POSIX ОС, Windows***	Файл, «спроецированный» в ОЗУ. Может изменяться путем переписывания содержимого памяти, а не записью в выходной поток

Кто синхронизует?

Что синхронизируется?

Как выполняется синхронизация?

Способы

Алгоритм синхронизации кода

СИНХРОНИЗАЦИЯ

Синхронизация: что?

- Внешние устройства
  - Одиночные / пулы однотипных
- Код
  - Параллельное выполнение в разных потоках одних участков исполнимых команд
  - Параллельное выполнение кода с ожиданием в точках синхронизации
- Память
  - Разделяемая память (между процессами)
  - Участки памяти процесса (между потоками)

Синхронизация: кто?

- ОС
  - На одном хосте
- Связующее ПО
  - На одном хосте
  - На разных хостах
- Сами устройства
  - Выступают в роли сервера

Синхронизация в жизни

Синхронизация: как? – 1/3

Что	Как
Внешние устройства	• ОС: Скрывает интерфейс к устройствам «внутри» себя (драйверы). Разные процессы (или один из нескольких потоков одного процесса) вызывают одновременно функции доступа, принадлежащие (являющиеся частью) ОС → Сводится к защите кода внутри ОС • Само устройство: Организует эксклюзивный доступ к себе по [сетевому] протоколу → Защита кода внутри ПО устройства → «Упаковывается» в драйвер в ОС клиента → см. ОС

Синхронизация: как? – 2/3

Что	Как
Код	Сами по себе команды [процессора] не могут быть прерваны при переключении контекста и в защите не нуждаются. Современные компьютеры не разделяют память команд и данных → Сводится к защите локальных данных функции Способы защиты: • Создать класс и вместо переменных теперь метода использовать переменные класса • Запретить начинать исполнять код функции, пока она не вернет значение в другом потоке → Запрет на продолжение исполнения кода до выполнения условия

Синхронизация: как? – 3/3

Что	Как
Память	• Можно [было бы] защищать каждую ячейку памяти → НО: Имеет смысл защищать только критически важные и, обычно, крупнее минимально адресуемых • Можно [было бы] разрешить помечать области памяти и блокировать выполнение команд манипуляции данными → НО: 1) Манипулирование происходит на уровне ЯП высокого уровня 2) Защита должна обеспечивать эксклюзивное выполнение групп команд, зависящих от семантики конкретной задачи → Используется защита от исполнения кода

Способы синхронизации – 1/3

Метод	Описание

Spinlock	• Создается флаг доступа
	• Перед получением доступа к защищенной области флаг проверяется:
	  • Если флаг «опущен», то его «поднимают» и выполнение продолжается
	  • Если флаг «поднят», то выполняется бесконечный цикл проверок до тех пор, пока флаг не опустится
	• После окончания работы с защищенной областью флаг опускается
	Проблема: расходуются ресурсы
	Преимущества: процесс/поток не засыпают

Способы синхронизации – 2/3

Метод	Описание
Barrier	• Описывается точка синхронизации и список синхронизуемых потоков • Каждый поток при достижении точки синхронизации уведомляет об этом систему: • Если все потоки достигли синхронизации выполнение продолжается • Если хоть один другой поток еще не достиг точки синхронизации, поток останавливается Недостаток: медленный поток тормозит остальных

Способы синхронизации – 3/3

Метод	Описание
Semaphore	Закрывает доступ к набору однотипных ресурсов. Вместо флага используется счетчик свободных ресурсов
Mutual exclusion	Бинарный флаг доступа к разделяемому ресурсу, аналог семафора со счетчиком, равным единице. В отличие от spinlock, приостанавливает выполнение потока до разблокировки ресурса

Синхронизация кода – 1/3

- Для обеспечения синхронизации используются синхронизирующие объекты (СО)
  - Создаются по инициативе потоков/процессов
  - Принадлежат ОС
  - Манипуляции ими производятся командами API ОС, выполняющимися в режиме ядра, то есть не прерываются при переключении контекстов

Синхронизация кода – 2/3

- Выделяют следующие операции с СО:
  - Создать СО
  - Получить ссылку на существующий СО, созданный другим потоком/процессом
  - Уничтожить СО
  - Проверить свободен ли СО (но не захватывать) – неблокирующий вызов
  - Проверить и захватить СО – блокирующий вызов
  - Освободить СО, захваченный ранее

Синхронизация кода (успех) – 3/3