1. Системное программирование: определение, назначение, применение.
2. Фреймворк операционной системы: определение, назначение, применение, состав. Стандарт POSIX.

Фреймворк операционной системы – набор библиотек и правила работы с ними, которые являются посредниками между разрабатываемым ПО и ядром ОС.

**Назначение** – упрощение процесса разработки(«используем общие функции»), обеспечивает базовые функции для разработки приложений(«управление процессами, памятью, файловой системой, сетью, обработка прерываний, правами доступа, энергопотребление, взаимодействие с пользователем»).

**Применение** – базовые дествия(«перечислены в Назначение»), на основе базовых действий создаются мобильные/настольные приложения(«Игры, СУБД, VCS, IDE, GUI, Драйверы»). Также инструменты для работы с графикой, базами данных.

**Состав** – API для («перечислены в Назначение»).

**Пример -** примером фреймворка операционной системы может служить .NET Framework от Microsoft - предоставляет набор библиотек и инструментов для разработки приложений под операционную систему Windows.

**Отличие от библиотеки** - Библиотека может быть использована в программном продукте просто как набор подпрограмм похожей функциональности, не влияя на архитектуру про-граммного продукта и не накладывая на неё никаких ограничений.

В то время как фреймворк диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию - «каркас», который нужно будет расширять и изменять согласно указанным требованиям

**POSIX(Portable Operation System Interface)** – набор стандартов, описывающих:

* 1. Интерфейсы между ОС и прикладной программой(«стандартизация системных вызовов, которые программы использую для взаимодействия с ОС»)
  2. Библиотеку языка С(«stdio.h – printf/scanf, stdlib.h – malloc/free, fcntl.h – open/write/close»)
  3. Набор приложений и их интерфейсов(«Пример – Shell. Стандартная командная оболочка. Стандартные утилиты командной строки: ls, cp, mv, rm»)

**Назначение** – обеспечивает совместимость различных UNIX – подобных ОС, переносимость программ на уровне исходного кода.

**UNIX** – Uniplexed Information and Computing Service(однозначная). Linux, Mac.

Многозадачность, многопользовательность, многопроцессорность, иерархическая файловая система, командная строка, стандарты и спецификации, сетевые возможности.

1. Применение процессов в ОС Windows, API для работы с процессами.

**Применение** - создании абстракции, которая позволяет осуществлять (псевдо – если одно-ядерный процессор) параллельные операции, даже если в системе присутствует всего один центральный процессор.

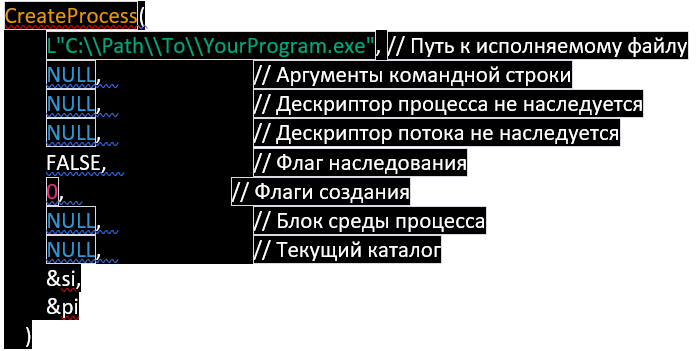
Процессы превращают один центральный процессор в несколько виртуальных – эффективное управление многозадачностью.

Процессы в многозадачной системе позволяют центральному процессору быстро переключаться между различными процессами, предоставляя каждому из них долю процессорного времени. Хотя в каждый конкретный момент времени процессор работает только с одним процессом, за секунду он может успеть поработать с несколькими, создавая иллюзию параллельной работы.

В случае многоядерных процессоров, несколько процессов могут действительно выполняться одновременно, поскольку каждое ядро может выполнять один процесс за раз.

**API:   
 CreateProcess**, которая создает новый процесс с единственным потоком. При вызове этой функции требуется указать имя файла исполняемой программы. Функция возвращает два отдельных дескриптора, по одному для процесса и потока, передавая их в структуре, которая указывается при вызове функции.

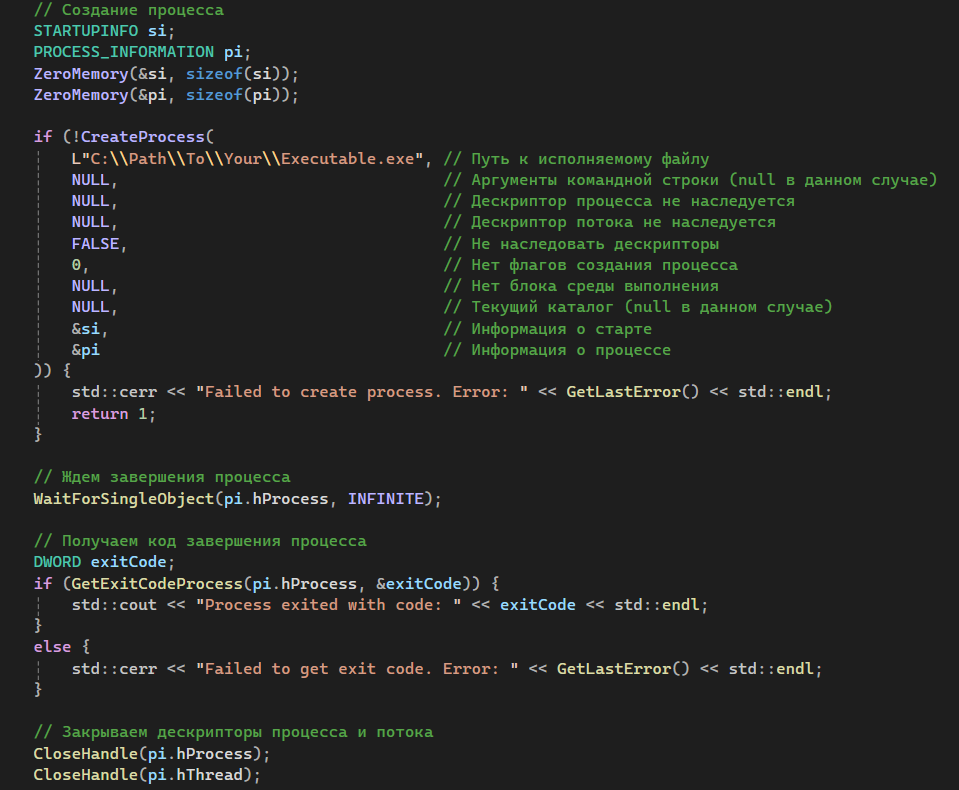
В Windows: одним вызовом функции Win32 CreateProcess создается процесс, и в него загружается нужная программа. У этого вызова имеется 10 парамет-ров, включая выполняемую программу, параметры командной строки для этой программы, различные параметры безопасности, биты, управляющие наследованием открытых файлов, информацию о приоритетах, спецификацию окна, создаваемого для процесса (если оно используется), и указатель на структуру, в которой вызывающей программе будет возвращена информация о только что созданном процессе



2 последний startupInfo и processInfo(информация о запущенном процессе)

**TerminateProcess(taskkill)** - остановить процесс в Windows.

**GetExitCodeProcess** - получать код завершения процесса соответственно



**4 события создания процесса**

1. Инициализация системы
2. Выполнение работающим процессом системного вызова, предназна-ченного для создания процесса
3. Запрос пользователя на создание нового процесса
4. Инициация пакетного задания

**Процесс** – единица работы ОС(объект ядра ОС + контекст процесса).

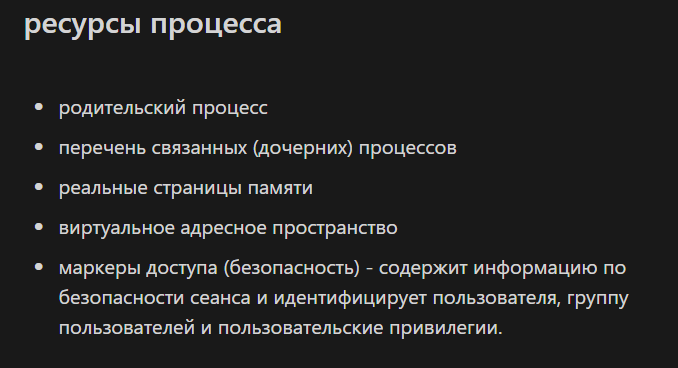
**Свойства:**

* 1. Процессу соответствует исполняемый файл
  2. PID
  3. PPID
  4. Перечень связанных дочерних процессов
  5. Приоритет(0-31)
  6. **Состояния:**
     1. New – создан, но не начал выполнение
     2. Ready – ожидает выделения процессорного времени для выполнения, находясь в очереди готовых процессов
     3. Running – выполняется на процессоре
     4. Blocked(Waiting) – временно приостановлен и ожидает события(«ввод/вывод, завершение работы другого процесса»).
     5. Terminated – завершен и освобождает выделенные ресурсы(«Конечное состояние»)
     6. Suspended – временно приостановлен, состояние сохранено – может быть возобновлен позже
  7. Атрибуты безопасности
  8. Handle – 32 – разрядный номер объекта ядра ОС.
  9. Запуск и управление процессом – системные вызовы
  10. Процессы изолированы друг от друга
  11. **Процессы выделено линейное адресное пространство.** **Сегменты:**
      1. Code – исполняемы код программы
      2. Data – глобальные переменные
      3. Heap – для динамического выделения памяти во время выполнения
      4. Stack – локальные переменные и аргументы функций
  12. **Стандартные потоки ввода/вывода -** процессу автоматически доступны 3 потока: ввода, вывода, вывод ошибок.
  13. **Системные процессы –** процессы, запускаемые автоматически при запуске OS; Windows: windows-сервисы; Linux-демоны. Системные процессы являются частью ядра и всегда расположены в оперативной памяти**.**
  14. **Пользовательские процессы -** процессы, запускаемые пользователем, выполняют собственный код и иногда обращаются к системным функциям.
  15. **Idle process -** процесс, выполняемый процессором в пространстве ядра операционной системы в случае, если нет других процессов, которые процессор мог бы выполнять.

**Контекст процесса** - данные, которые сохраняются при переключении процессов и предназначенные для продолжения работы

**Ресурсы**

Процесс предоставляет ресурсы, необходимые для выполнения программы. Процесс имеет виртуальное адресное пространство, исполняемый код, открытые дескрипторы для системных объектов, контекст безопасности, уникальный идентификатор процесса, переменные среды, класс приоритета, минимальный и максимальный размер рабочего набора и по крайней мере один поток выполнения



1. Методы межпроцессного взаимодействия в ОС Windows: обмен данными, синхронизация.

**Обмен данными (механизмы IPC (Inter-Process Communication)):**

1. **Именованный канал** - это один из методов межпроцессного взаимодействия, который обеспечивает однонаправленную или двунаправленную связь между сервером канала и одним или несколькими клиентами канала1. Все экземпляры именованного канала используют одно и то же имя канала, но каждый эк-земпляр имеет свои собственные буферы и дескрипторы и предоставляет от-дельный канал для связи клиент/сервер2. Именованные каналы поддерживают полнодуплексную связь по сети и несколько экземпляров сервера, обмен сообщениями и имитацию клиента, что позволяет подключающимся процессам использовать свой собственный набор разрешений на удаленных серверах.
2. **Сокеты** - механизм для обмена данными между процессами через сетевые соединения. TCP/IP сокеты могут использоваться для взаимодействия по локальной сети или в Интернете.
3. **Mailslot** - механизм для однонаправленного межпроцессного взаимодействия. Приложения могут хранить сообщения в Mailslot. Владелец Mailslot может из-влекать сообщения, которые там хранятся8. Эти сообщения обычно передают-ся по сети либо на указанный компьютер, либо на все компьютеры в указан-ном домене8. Mailslot представляет собой серверно-клиентский интер-фейс. Сервер может создать Mailslot, а клиент может записать в него по имени. Только сервер может читать Mailslot, поэтому Mailslot представляет собой механизм одностороннего взаимодействия.
4. **Очереди сообщений** - Очереди сообщений позволяют процессам отправлять и получать сообщения в асинхронном режиме.
5. **Memory-Mapped Files(Shared Memory)** - позволяют нескольким процессам разделять одинаковую область памяти.

**Синхронизация(целостность какого-либо ресурса (файл, данные в памяти), когда он используется несколькими процессами или потоками в случайном порядке):**

1. Семафоры: Семафоры используются для контроля доступа к общим ресурсам. Они представляют собой переменную, которая показывает, сколько ресурсов доступно для использования.
2. Мьютексы: Мьютексы, или взаимные исключения, используются для предотвращения одновременного доступа к общему ресурсу. Если один процесс использует ресурс, другие процессы должны ждать, пока первый процесс не освободит ресурс.
3. События: События используются для уведомления одного или нескольких процессов о том, что произошло определенное событие.

**Общее:**

1. **RPC:** вызывать функции или процедуры в удаленных процессах, обеспечивая передачу аргументов и результатов.

Механизмы синхронизации - удаленные мьютексы

1. Применение потоков в ОС Windows, API для работы с потоками, API для синхронизации потоков.

**Поток** - объект ядра операционной системы, которому OS выделяет про-цессорное время. Наименьшая единица работы ядра OS.

**Поток** — это сущность в процессе, которую можно запланировать для выполнения. Все потоки процесса совместно используют его виртуальное адресное пространство и системные ресурсы

1. Применение механизмов синхронизации в ОС Windows, API для синхронизации.

1. Файловая система: логическая и физическая организация данных, определение файловой системы, отличие файловых систем, оглавление файловой системы, файлы, каталоги, основные функции файловой системы, буферы ввода/вывода, кеширование ввода/вывода, основные функции API файловой системы, маркер файла, текущая позиция файла, блокировка файлов, наблюдение за изменением в каталоге, особенности устройства файловой системы в Linux.
2. Работа с оперативной памятью в ОС Windows: API для работы с виртуальной памятью, API для работы с Heap.
3. Механизм отображение файлов в памяти: последовательность системных вызовов Windows для создания образа файла в оперативной памяти, использование образа файла, как средства межпроцессного взаимодействия.
4. Динамически вызываемые библиотеки: структура DLL-библиотеки, экспорт функций, загрузка динамической библиотеки, динамический вызов функций динамической библиотеки, создание и применение библиотеки импорта.
5. Спецификация COM: понятие позднего связывания программных модулей, COM-интерфейс, стандартные COM-интерфейсы, структура COM-клиента, структура COM/DLL-сервера, экспортируемые стандартные функции, регистрация COM/DLL-сервера.
6. Управление пользователями и группами пользователей в Windows: понятие дискреционной системы безопасности, типы Windows-пользователей, группы пользователей, возможности API управления пользователями и группами.

1. Структурная обработка ошибок в Windows: программное исключение, программные конструкции для обработки ошибок в Windows, фильтры, возможности API для структурной обработки ошибок, генерация ошибок, финальная обработка исключений.
2. Windows-консоль: определение, применение стандартных потоков для ввода/вывода в консоль, возможности API для управления консолью.
3. Windows-сервисы: определение, назначение, применение, API.
4. Асинхронные операции ввода вывода: понятие асинхронной операции ввода/вывода, особенности программирования асинхронного ввода/вывода.
5. Порты завершения ввода/вывода: назначение, применение, API.
6. Платформа Docker: архитектура, назначение, принципы устройства, файловая система UFS, контейнеры, образы, основные команды.