
OC

Что такое ОС?

ОС (Операционная система) - программное обеспечение (ПО), управляющее ресурсами компьютера (процессор, память, ввод/вывод) и обеспечивающее взаимодействие с приложениями. Пример: **Windows, Linux, macOS.**




Координирует работу программ, обеспечивает интерфейс для ввода команд (использует графический интерфейс или командную строку).



Терминал (terminal) - интерфейс для взаимодействия с ОС. Пример: командная строка в терминале **Linux** или PowerShell/CMD в **Windows**.

Основные команды (linux):

- **ls** - отобразить содержимое текущей директории.
- **cd** - изменить текущую директорию.
- **pwd** - вывести полный путь текущей директории.
- **cp** - скопировать файл или директорию.
- **mv** - переместить или переименовать файл или директорию.
- **rm** - удалить файл или директорию.
- **mkdir** - создать новую директорию.
- **rmdir** - удалить пустую директорию.
- **cat** - вывести содержимое файла.
- **man** - получить справочную информацию о команде.
- **ps** - отобразить текущие процессы.
- **kill** - завершить процесс. (kill id)

```
TERMINAL  bash - hello-world + ~   ^ >
```

```
~ took 508ms
➤ mkdir hello-world && cd hello-world

~/hello-world
➤ git init
Initialized empty Git repository in /Users/daimms/hello-world/.git/

~/hello-world  main
➤ echo "test" > test_file

~/hello-world  main  ?
➤ git add . && git commit -m "Hello world!"
[main (root-commit) c1c1283] Hello world!
 1 file changed, 1 insertion(+)
 create mode 100644 test_file

~/hello-world  main
➤
```

МОДУЛЬ os:
`import os`

Что такое процесс?

Процесс — это выполняемая программа вместе с ее текущим состоянием. Процесс включает:

1. **Код программы:** Инструкции, которые выполняет процессор.
2. **Данные программы:** Переменные, структуры данных и т.д.
3. **Контекст выполнения:** Состояние регистров процессора, счетчик команд и т.д.
4. **Системные ресурсы:** Открытые файлы, каналы, сигналы и т.д.

Создание процесса

В Unix-подобных системах процессы обычно создаются с помощью системного вызова `fork()`. Этот вызов создает новый процесс, копируя текущий процесс. Новый процесс называется дочерним, а исходный процесс — родительским.

```
pid = os.fork()
```

PID;

PID (Process ID) — уникальный идентификатор процесса в системе. С его помощью операционная система и приложения могут различать и управлять процессами. Каждый процесс в системе имеет свой PID.

- **Родительский процесс:** Процесс, который создал текущий процесс. Его PID можно получить с помощью `os.getppid()`.
- **Дочерний процесс:** Процесс, созданный другим процессом.

```
if pid > 0:
    # Родительский процесс
    print(f'parent')
else:
    # Дочерний процесс
    print(f'child')
```

Основные системные вызовы, ф-ии

os.fork():

- Создает новый процесс, копируя текущий процесс.
- В родительском процессе возвращает PID дочернего процесса, в дочернем — 0.

os.getpid():

- Возвращает PID текущего процесса.

os.getppid():

- Возвращает PID родительского процесса.

os.exec():

- Заменяет текущий процесс новым процессом, загружая и выполняя новую программу.
- Примеры: os.execl(), os.execsp(), os.execvp() и другие.

os.wait():

- Ожидает завершения дочернего процесса и возвращает его PID и код завершения.

os.waitpid(pid, options):

- Ожидает завершения конкретного дочернего процесса с указанным PID.
-

Процессы - пример:

```
import os

pid = os.fork()

if pid > 0:
    # Родительский процесс
    pid, status = os.wait()
    print(f'Child {pid} exited with status {status}')
else:
    # Дочерний процесс
    print('Child process is running')
    os._exit(0) # Завершение дочернего процесса
```

Понятие сигнала.

Сигнал — это механизм межпроцессного взаимодействия в Unix-подобных операционных системах. Сигналы используются для уведомления процесса о каком-либо событии. Они могут быть отправлены процессу ядром операционной системы, другим процессом или самим процессом. Используется специализированная библиотека/модуль `import signal`;

SIGINT (2):

- Посылается при прерывании программы пользователем (обычно нажатием Ctrl+C).
- Действие по умолчанию: завершение процесса.

SIGTERM (15):

- Посылается для запроса завершения процесса.
- Действие по умолчанию: завершение процесса, но может быть перехвачен и обработан.

SIGKILL (9):

- Немедленное завершение процесса. Этот сигнал не может быть перехвачен или проигнорирован.
 - Действие по умолчанию: завершение процесса.
-

«продолжение сигналов»

SIGHUP (1):

- Посылается при разрыве управляющего терминала или завершении процесса, который его контролирует.
- Действие по умолчанию: завершение процесса, но часто используется для перезагрузки конфигурационных файлов.

SIGQUIT (3):

- Посылается при нажатии Ctrl+\. Создает дамп памяти и завершает процесс.
- Действие по умолчанию: завершение процесса с дампом памяти.

SIGALRM (14):

- Посылается таймером, установленным функцией alarm().
- Действие по умолчанию: завершение процесса, но может быть перехвачен и обработан.

SIGCHLD (17):

- Посылается родительскому процессу при завершении или остановке дочернего процесса.
 - Действие по умолчанию: игнорирование.
-

«продолжение сигналов»

SIGSTOP (19):

- Немедленная остановка процесса. Этот сигнал не может быть перехвачен или проигнорирован.
- Действие по умолчанию: остановка процесса.

SIGCONT (18):

- Возобновление выполнения остановленного процесса.
- Действие по умолчанию: продолжение выполнения процесса.

SIGUSR1 (10) и SIGUSR2 (12):

- Пользовательские сигналы, предназначенные для произвольного использования.
 - Действие по умолчанию: завершение процесса, но могут быть перехвачены и обработаны.
-

УНИЧТОЖЕНИЕ ПРОЦЕССА!!!

Команда kill:

- Посылает сигнал процессу для его завершения.
- Синтаксис: kill [signal] PID
- Пример: kill -9 1234 (посылает сигнал SIGKILL процессу с PID 1234, немедленно завершив его).

```
pid = 1234 # PID процесса, который нужно завершить  
os.kill(pid, signal.SIGKILL)
```

обработчик сигнала;

Обработчик сигнала — это функция, которая вызывается, когда процесс получает сигнал. Обработчики сигналов позволяют процессу реагировать на сигналы различными способами, например, выполняя очистку перед завершением, игнорируя сигнал или выполняя специальные действия.

пример - обработка сигналов;

```
import signal
import time

# Функция-обработчик сигнала
def handle_signal(signum, frame):
    print(f'Received signal {signum}')

# Установка обработчиков сигналов
signal.signal(signal.SIGINT, handle_signal)
signal.signal(signal.SIGTERM, handle_signal)

print('Press Ctrl+C to send SIGINT signal or send SIGTERM signal to the
process...')

try:
    while True:
        time.sleep(1) # Программа спит, ожидая сигналы
except KeyboardInterrupt:
    print('Exiting gracefully...')
```

Что такое поток?

Поток (Thread) — это наименьшая единица обработки, управляемая планировщиком операционной системы. Потоки позволяют выполнять несколько задач параллельно внутри одного процесса. Вот основные характеристики потоков:

1. **Разделение ресурсов:** Потоки внутри одного процесса разделяют память и ресурсы (файлы, данные и т.д.) этого процесса.
 2. **Параллельное выполнение:** Потоки позволяют выполнять несколько операций одновременно, что может улучшить производительность, особенно на многоядерных процессорах.
 3. **Легковесность:** Создание и переключение между потоками происходит быстрее и требует меньше ресурсов по сравнению с процессами.
-

общий пример - потоки;

```
import threading

def print_numbers():
    for i in range(5):
        print(i)

def print_letters():
    for letter in 'abcde':
        print(letter)

# Создание потоков
thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)

# Запуск потоков
thread1.start()
thread2.start()

# Ожидание завершения потоков
thread1.join()
thread2.join()
```

Управление процессами/памятью.

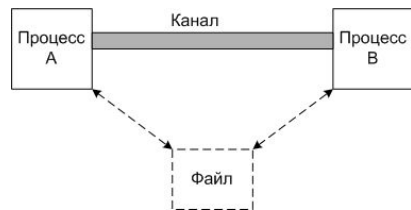
Управление процессами - ключевая функция ОС, обеспечивающая контроль и координацию выполнения программ. ОС выделяет ресурсы, такие как процессорное время и оперативная память, для каждого процесса. Управление процессами включает в себя планирование выполнения, аллокацию ресурсов, обработку прерываний и контроль за состоянием процессов.

Пример: **Диспетчер задач в Windows** - инструмент, предоставляемый операционной системой для отслеживания и управления процессами. Он позволяет пользователю просматривать запущенные приложения, завершать процессы и управлять ресурсами системы.

Управление памятью - функция ОС ответственна за эффективное использование оперативной и внутренней памяти компьютера. Операционная система управляет выделением и освобождением памяти для процессов, обеспечивая оптимальное распределение ресурсов. Основные концепции включают в себя виртуальную память, страничное перемещение и управление кэшем.

Пример: **Виртуальная память в ОС** - механизм, позволяющий программам использовать больше оперативной памяти, чем физически доступно. Он использует комбинацию оперативной памяти и пространства на жестком диске для создания иллюзии большего объема памяти.

Межпроцессорное взаимодействие.



Межпроцессорное взаимодействие - это обмен данными и координация между несколькими процессами в компьютерной системе. Методы включают:

- **семафоры**, механизм, используемый для контроля доступа к общим ресурсам.
- **очереди сообщений**, процессы могут обмениваться сообщениями через специальные очереди.
- **разделяемую память**, создание общего участка в памяти
- **каналы связи и сигналы.**
- **каналы делятся на:**
 - **именованные**
 - **неименованные**

Применяется в **многозадачных системах, сетевых приложениях и параллельных вычислениях.**

Именованный канал;

Именованные каналы (Named Pipes)

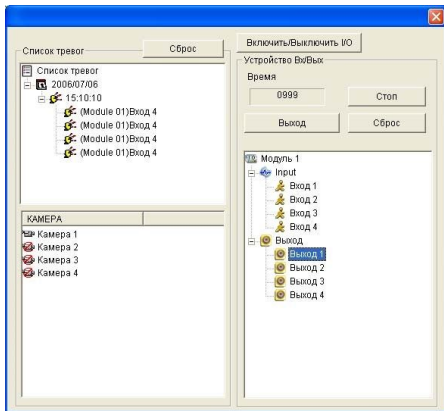
1. **Создаются с помощью команды `mkfifo` или функции `mkfifo()`:**
 - Пример: `os.mkfifo('/tmp/my_fifo')`
 2. **Обеспечивают двустороннюю связь:**
 - В отличие от неименованных каналов, можно организовать двустороннее общение через один именованный канал.
 3. **Могут использоваться для связи между любыми процессами:**
 - Любой процесс, имеющий доступ к именованному каналу, может читать из него и писать в него.
 4. **Имеют имя в файловой системе:**
 - Видны как файлы в файловой системе и могут быть открыты по имени.
-

Неименованный канал;

Неименованные каналы (Unnamed Pipes)

1. **Создаются с помощью функции `pipe()`:**
 - Пример: `pipe_fd = os.pipe()`
 2. **Обеспечивают одностороннюю связь:**
 - Одно направление: от одного процесса к другому.
 3. **Используются для связи между процессами, имеющими родительно-дочерние отношения:**
 - Обычно создаются в родительском процессе до создания дочернего процесса через `fork()`.
 4. **Видны только в пределах процесса, который их создал, и его дочерних процессов:**
 - Не имеют имени в файловой системе.
-

Управление I/O



Управление вводом/выводом (I/O) - важная функция операционной системы (ОС), ответственная за координацию взаимодействия компьютера с внешними устройствами ввода и вывода. Основными аспектами в управлении являются:

- **Драйверы устройств:** Программы для взаимодействия с устройствами.
- **Буферизация данных:** Временное хранение данных для эффективного обмена.
- **Контроль доступа:** Регулирование доступа к устройствам.

Управление I/O обеспечивает эффективный обмен данными между компьютером и внешними устройствами, такими как жесткие диски, сетевые устройства, принтеры и другие периферийные устройства.

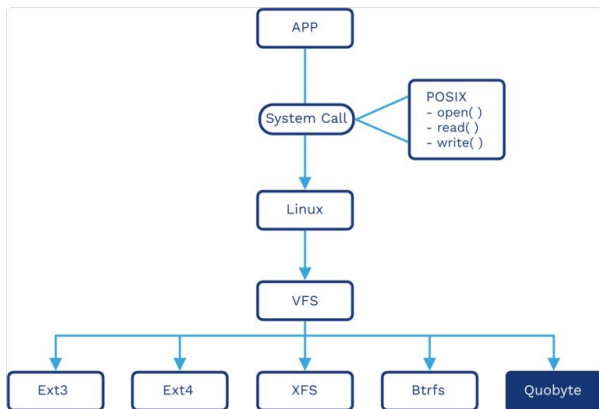
POSIX

POSIX (Portable Operating System Interface) - набор стандартов, разработанных для обеспечения совместимости между UNIX-подобными операционными системами. Эти стандарты определяют интерфейсы и API (Application Programming Interface), обеспечивая переносимость программного обеспечения между различными UNIX-подобными системами. Ключевыми аспектами являются:

- **Совместимость:** POSIX - стандарт для переносимости приложений.
 - **API:** Набор определенных функций для программирования в POSIX.
 - **Интерфейсы:** Устанавливает стандарты взаимодействия программ с ОС.
-

POSIX.

показан уровень иерархии файловой системы. Разбивка схемы:



- Пользовательское приложение (APP) выполняет системный вызов (например, `open()`, `read()` или `write()`).
- Системный вызов обрабатывается ядром Linux, которое обеспечивает уровень абстракции, называемый
- VFS предоставляет общий интерфейс для различных файловых систем, таких как Ext3, Ext4, XFS, Btrfs и Quobyte.
- Каждая файловая система имеет свою собственную реализацию и структуру данных

На схеме POSIX показан как стандарт, определяющий интерфейс между пользовательским приложением и файловой системой. Это гарантирует, что приложение может отправлять запросы на файловые операции, такие как открытие, чтение и запись, а файловая система может понимать и выполнять эти запросы, независимо от конкретной используемой файловой системы.

Придерживаясь стандарта POSIX, различные файловые системы можно использовать взаимозаменяемо, что обеспечивает гибкость и переносимость приложений, поскольку их можно написать один раз и запускать в разных операционных и файловых системах.

POSIX. Примеры.

POSIX.1: Основные стандарты, включающие стандартные библиотеки, командные интерфейсы и общие утилиты.

POSIX.2: Дополнительные командные интерфейсы и утилиты, включая команды для работы с процессами и файловыми системами.

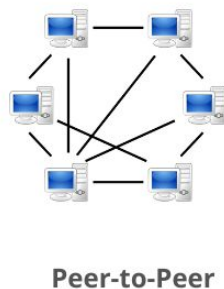
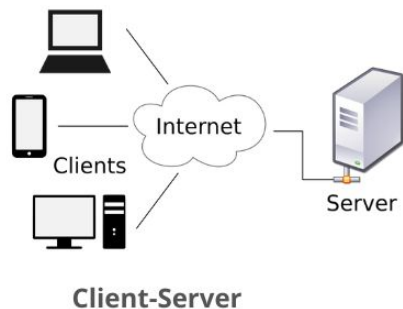
POSIX.4: Стандарт для реального времени, определяющий интерфейсы для управления временем и сигналами в реальном времени.

Концепции построения сетей.

Главные аспекты построения сетей:

- **Архитектура** - физическая и логическая топология, например, звезда, кольцо, TCP/IP.
 - **Протоколы** - правила обмена данными, как TCP/IP, HTTP, Ethernet.
 - **Безопасность** - методы защиты от доступа, атак и утечек, включая шифрование и брандмауэры.
-

«продолжение» концепц.постр. сетей



- **Client-Server (Клиент-Сервер):** Сетевая архитектура, где клиенты запрашивают ресурсы у сервера. Пример: веб-сервер и браузеры.
 - **Peer-to-Peer (Пиринговая сеть):** Компьютеры обмениваются ресурсами напрямую без центрального сервера. Пример: файлообменные сети.
 - **TCP/IP:** Протокольная семья, используемая в интернете для передачи данных между устройствами.
 - **Firewall (Брандмауэр):** Средство безопасности, контролирующее трафик между сетями и обеспечивающее защиту от несанкционированного доступа.
-

multiprocessing

Модуль multiprocessing предоставляет средства для параллельного выполнения задач с помощью создания и управления процессами. Он позволяет создавать процессы, аналогичные потокам, но с отдельными пространствами памяти, что позволяет эффективно использовать многопроцессорные системы.

Создание процесса:

- Для создания процесса используется класс `multiprocessing.Process`.
- Основные параметры конструктора:
 - `target`: Функция, которую процесс должен выполнять.
 - `args`: Аргументы, передаваемые функции.

Запуск и завершение процесса:

- `start()`: Запуск процесса.
 - `join()`: Ожидание завершения процесса.
 - `terminate()`: Принудительное завершение процесса
-

ПЛЮСЫ И МИНУСЫ;

Преимущества:

- Полное разделение памяти между процессами предотвращает проблемы, связанные с конкурентным доступом.
- Полная независимость процессов позволяет избежать ошибок, которые могут возникать при работе с потоками.

Ограничения:

- Создание процессов требует больших накладных расходов по сравнению с потоками.
 - Обмен данными между процессами может быть медленнее из-за необходимости межпроцессного взаимодействия.
-

пример-1:

```
import multiprocessing

def worker():
    print('Worker function is running')

if __name__ == '__main__':
    process = multiprocessing.Process(target=worker)
    process.start()
    process.join()    # Ожидание завершения процесса
```

пример-2:

```
import multiprocessing
import time

def worker():
    while True:
        print('Working...')
        time.sleep(1)

if __name__ == '__main__':
    process = multiprocessing.Process(target=worker)
    process.start()
    time.sleep(5)
    process.terminate()
    process.join()
    print('Process has been terminated')
```

Задачи ОС.

1. написать программу, которая создает два потока: 1 поток спит 10 сек и завершается; 2 поток выводит идентификатор процесса и завершается.
 2. написать программу, которая создает новый процесс и уничтожает родителя сигналом .
 3. написать программу, которая создает два потока. Первый поток принимает строку и выводит её, второй поток принимает целое число, засыпает на указанное количество секунд, затем завершается.
 4. <на **multiprocessing**> напишите программу, которая создает два процесса. Один процесс будет печатать "Hello" 5 раз, а другой процесс будет печатать "World" 5 раз.
 5. <на **multiprocessing**> напишите программу, которая создает два процесса. Первый процесс будет вычислять сумму чисел от 1 до 100 и выводить результат. Второй процесс будет вычислять произведение чисел от 1 до 10 и выводить результат. Основной процесс должен дожидаться завершения обоих процессов и затем напечатать сообщение о завершении работы.
 6. напишите программу, которая создает два потока. Один поток будет печатать числа от 1 до 5, а другой поток будет печатать буквы от 'a' до 'e'.
-