Работа с сетью, сокеты

- Что такое сокеты?
- Зачем нужны сокеты?
- Программа клиент-сервер

In [ ]:

При помощи сокетов можно организовать взаимодействие между процессами,
работающим на разных серверах.

Сокет в ОС Linux - это объект уровня ядра.
Т.е. python процесс, при создании сокета и вызова функций осуществяет системные вызовы.
Ядро ОС возвращает результаты работы системных вызовов python процессу.

In [ ]:

```
# создание сокета, сервер

import socket

# https://docs.python.org/3/library/socket.html
sock=socket.socket(socket.AF_INET, socket.SOCK_STR
EAM)
sock.bind(("127.0.0.1", 10001))   # max port 65535
sock.listen(socket.SOMAXCONN)

conn, addr = sock.accept()
while True:
    data = conn.recv(1024)
    if not data:
        break
    # process data
    print(data.decode("utf8"))

conn.close()
sock.close()
```

In [ ]:

```
# создание сокета, клиент

import socket

sock = socket.socket()
sock.connect(("127.0.0.1", 10001))
sock.sendall("ping".encode("utf8"))
sock.close()

# более короткая запись

sock = socket.create_connection(("127.0.0.1", 1000
1))
sock.sendall("ping".encode("utf8"))
sock.close()
```

In [ ]:

```
# создание сокета, контекстный менеджер
# сервер
import socket

with socket.socket() as sock:
    sock.bind(("", 10001))
    sock.listen()

    while True:
        conn, addr = sock.accept()
        with conn:
            while True:
                data = conn.recv(1024)
                if not data:
                    break
                print(data.decode("utf8"))

# клиент
import socket

with socket.create_connection(("127.0.0.1", 10001)
) as sock:
    sock.sendall("ping".encode("utf8"))
```

Таймауты и обработка сетевых ошибок

- connect timeout и read timeout, в чем разница?
- обработка ошибок

In [ ]:

```
# создание сокета, таймауты и обработка ошибок
# сервер
import socket

with socket.socket() as sock:
    sock.bind(("", 10001))
    sock.listen()
    while True:
        conn, addr = sock.accept()
        conn.settimeout(5)  # timeout := None|0|gt
0
        with conn:
            while True:
                try:
                    data = conn.recv(1024)
                except socket.timeout:
                    print("close connection by tim
eout")
                    break

                if not data:
                    break
                print(data.decode("utf8"))
```

In [ ]:
```
# создание сокета, таймауты и обработка ошибок
# клиент
import socket

with socket.create_connection(("127.0.0.1", 10001)
, 5) as sock:
    # set socket read timeout
    sock.settimeout(2)
    try:
        sock.sendall("ping".encode("utf8"))
    except socket.timeout:
        print("send data timeout")
    except socket.error as ex:
        print("send data error:", ex)
```

Одновременная обработка нескольких соединений

- Как обработать несколько соединений одновременно?
- Что использовать процессы или потоки?
- Рассмотрим примеры обработки сетевых запросов

In [ ]:
```
# обработка нескольких соединений одновременно

import socket

with socket.socket() as sock:
    sock.bind(("", 10001))
    sock.listen()
    while True:
        conn, addr = sock.accept()
        print("connected client:", addr)
        # процесс или поток для обработки соединения
        with conn:
            while True:
                data = conn.recv(1024)
                if not data:
                    break
                print(data.decode("utf8"))
```

In [ ]:
Какие преимущества дает создание процессов?
Можно утилизировать все ядра CPU.
Вызов fork - это слишком тяжелая операция.
Иногда дороже сделать fork, чем обработать сам запрос.
Для процессов будет большой расход памяти.

Использование потоков ограничено GIL и одним процессом.
Мы можем исчерпать 100% CPU на одном ядре, программа будет работать неэффективно.

Как поведет себя ОС при большом кол-ве процессов или потоков?

In [ ]:

```
# обработка нескольких соединений одновременно, потоки

import socket
import threading

def process_request(conn, addr):
    print("connected client:", addr)
    with conn:
        while True:
            data = conn.recv(1024)
            if not data:
                break
            print(data.decode("utf8"))

with socket.socket() as sock:
    sock.bind(("", 10001))
    sock.listen()
    while True:
        conn, addr = sock.accept()
        th = threading.Thread(target=process_reque
st, args=(conn, addr))
        th.start()
```

In [ ]:

```
# обработка нескольких соединений одновременно, процессы и потоки
import socket

with socket.socket() as sock:
    sock.bind(("", 10001))
    sock.listen()
    # создание нескольких процессов
    while True:
        # accept распределится "равномерно" между процессами
        conn, addr = sock.accept()
        # поток для обработки соединения
        with conn:
            while True:
                data = conn.recv(1024)
                if not data:
                    break
                print(data.decode("utf8"))
```

In [ ]:

```
# обработка нескольких соединений одновременно, процессы и потоки

import socket
import threading
import multiprocessing

with socket.socket() as sock:
    sock.bind(("", 10001))
    sock.listen()

    workers_count = 3
    workers_list = [multiprocessing.Process(target
=worker, args=(sock,))
                    for _ in range(workers_count)]

    for w in workers_list:
        w.start()

    for w in workers_list:
        w.join()
```

In [ ]:

```
# обработка нескольких соединений одновременно, процессы и потоки

def worker(sock):
    while True:
        conn, addr = sock.accept()
        print("pid", os.getpid())
        th = threading.Thread(target=process_request, args=(conn, addr))
        th.start()

def process_request(conn, addr):
    print("connected client:", addr)
    with conn:
        while True:
            data = conn.recv(1024)
            if not data:
                break
            print(data.decode("utf8"))
```