Web сервера


NGINX
http server


APACHE
HTTP SERVER


LIGHTTPD
fly light.


IIS7
internet information services

Запуск web сервера

- Команда на запуск

```
sudo /etc/init.d/nginx start
```

- Чтение файла конфигурации
- Получение порта 80
- Открытие (создание) логов
- Понижение привилегий
- Запуск дочерних процессов/потоков (*)
- Готов к обработке запроса

Файлы web сервера

- Конфиг `/etc/nginx/nginx.conf`

  `include /etc/nginx/sites-enabled/*`
- Init-скрипт `/etc/init.d/nginx [start|stop|restart]`
- PID-файл `/var/run/nginx.pid`
- Error-лог `/var/log/nginx/error.log`
- Access-лог `/var/log/nginx/access.log`

Процессы web сервера

- Master (root, 1 процесс)
  - Чтение и валидация конфига
  - Открытие сокета (ов) и логов
  - Запуск и управление дочерними процессами (worker)
  - Graceful restart, Binary updates
- Worker (www-data, 1+ процессов)
  - Обработка входящих запросов


Чтение HTTP запроса
Выбор virtual host
Выбор location
Проверка доступа
Чтение файла
Применение фильтров
Отправка HTTP ответа
Запись Access Log
Очистка

Модульная архитектура

- web сервер – не монолитный
- Динамическая загрузка модулей - LoadModule
- Этапы обработки запроса и модули
- Дополнительные директивы, контексты
- Примеры: mod_mime, mod_mime_magic, mod_autoindex, mod_rewrite, mod_cgi, mod_perl, mod_gzip

Конфигурация web сервера

Терминология

virtual host, **вирт. хост** - секция когфига web сервера, отвечающая за обслуживание определенного домена

location - секция когфига, отвечающая за обслуживание определенной группы URL

```
user  www www;
error_log  /var/log/nginx.error_log  info;
http {
  include       conf/mime.types;
  default_type  application/octet-stream;
  log_format    simple '$remote_addr $request $status';
  server {
    listen        one.example.com;
    server_name   one.example.com www.one.example.com;
    access_log    /var/log/nginx.access_log simple;
    location / {
      root        /www/one.example.com;
    }
    location ~* ^.+\.(jpg|jpeg|gif)$ {
      root        /www/images;
      access_log  off;
      expires     30d;
    }
  }
}
```

Секции и директивы

- `http` — конфигурация для HTTP сервера
- `server` — конфигурация домена (вирт. Хоста)
- `server_name` — имена доменов
- `location` — локейшен, группа URL
- `root`, `alias` — откуда нужно брать файлы
- `error_log` — лог ошибок сервера
- `access_log` — лог запросов

Приоритеты location в nginx

- `location = /img/1.jpg`
- `location ^~ /pic/`
- `location ~* \.jpg$`
- `location /img/`

При одинаковом приоритете используется тот location, что находится **выше** в конфиге.

Отдача статических документов

```
location ~* ^.+\.(jpg|jpeg|gif)$ {
    root         /www/images;
}
location /sitemap/ {
    alias /home/www/generated;
}
```

`/2015/10/ae2b5.png` → `/www/images/2015/10/ae2b5.png`

`/sitemap/index.xml` → `/home/www/generated/index.xml`

Аттрибуты файлов и процессов

У процесса есть

- пользователь
- группа

У файла (или директории) есть

- пользователь (владелец)
- группа
- права доступа (read/write/execute)

Как узнать атрибуты ?

```
$ ps -o pid,euser,egroup,comm,args -C nginx
  PID EUSER    EGROUP   COMMAND
29731 root     root     nginx: master process /usr/sbin/nginx
29732 www-data www-data nginx: worker process
29733 www-data www-data nginx: worker process
29734 www-data www-data nginx: worker process
29737 www-data www-data nginx: worker process

$ ls -lah www/index.html
-rw-r--r-- 1 nuf users 156K Feb  6 21:15 www/index.html
```

Проверка доступа

Для того что бы открыть файл - необходимо иметь права на чтение `r` самого файла и на исполнение `x` директорий, в которых он находится. Наличие прав проверяется следующим образом:

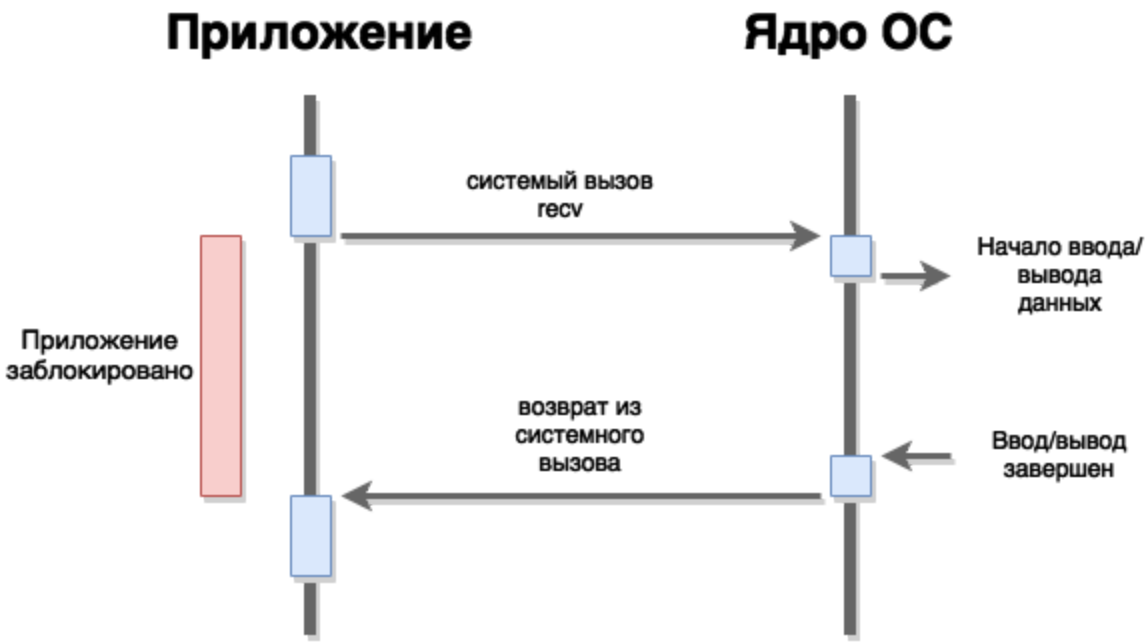
- Если совпадает пользователь `-rw-r--r--`
- Если совпадает группа `-rw-r--r--`
- Иначе `-rw-r--r--`

Модели обработки сетевых соединений

Простейший TCP сервер

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('127.0.0.1', 8080))
s.listen(10)
while True:
    conn, addr = s.accept()
    path = conn.recv(512).decode('utf8').rstrip("\r\n")
    file = open('/www' + str(path), 'r')
    data = file.read().encode('utf8')
    conn.sendall(data)
    file.close(); conn.close()
```

Блокирующий ввод-вывод



Решение проблемы

- множество потоков - multithreading
- множество процессов - prefork, pool of workers
- комбинированный подход

Плюсы и минусы prefork

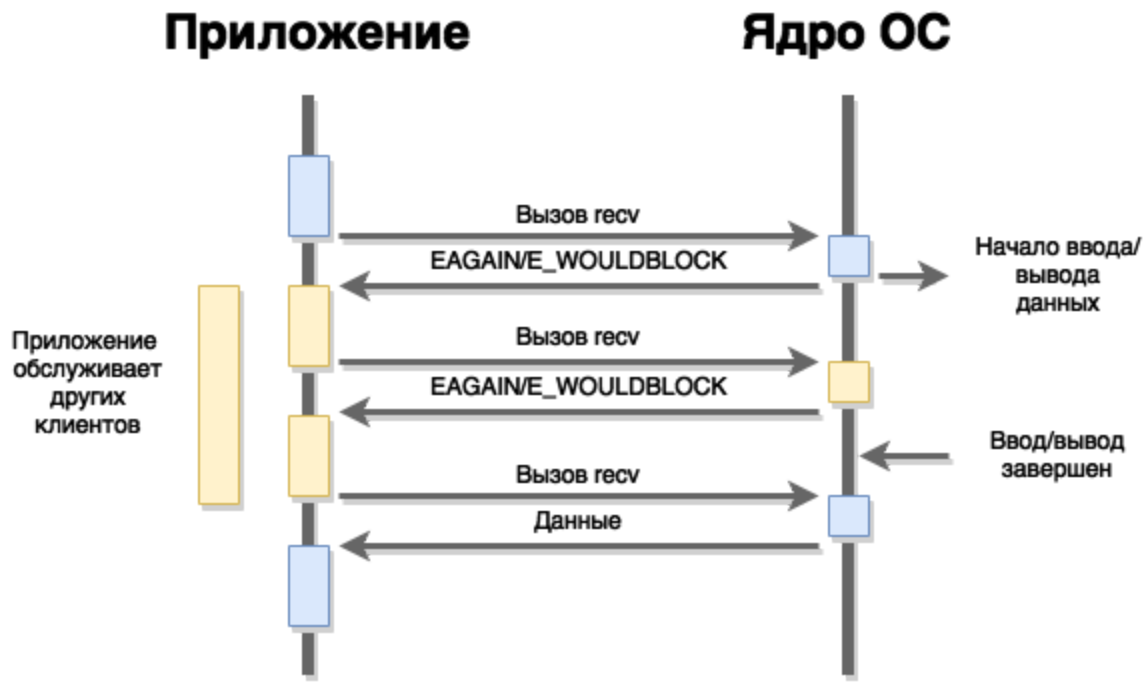
+ простота разработки
+ можно использовать любые библиотеки
− большое потребление памяти: 1 клиент = 1 процесс
− проблема с долгоживущими соединениями

Плюсы и минусы multithreading

По сравнению с prefork,

+ экономия памяти: 1 клиент = 1 поток
− требует аккуратной работы с памятью
− как следствие, накладывает ограничение на выбор библиотек

Неблокирующий ввод-вывод



Мультиплексирование

```
readsocks, writesocks = [...], [...] # сокеты
while True:
    readables, writeables, exceptions = \
        select(readsocks, writesocks, [])
    for sockobj in readables:
        data = sockobj.recv(512)
        if not data:
            sockobj.close()
            readsocks.remove(sockobj)
        else:
            print('\tgot', data, 'on', id(sockobj))
```

Event-driven разработка

- множество открытых файлов
- select, kqueue, epoll, aio…
- последовательное исполнение → события

Плюсы и минусы

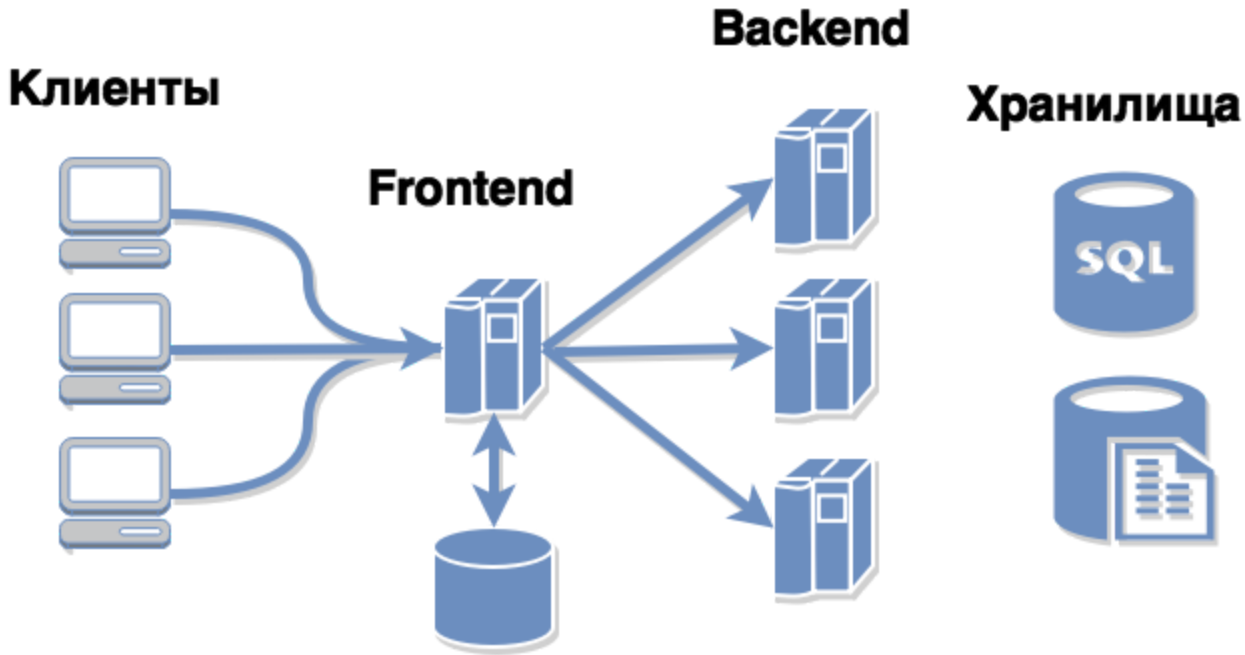
+ быстро, программа не блокируется
+ экономия памяти: 1 клиент = 1 объект
+ обработка большого количества клиентов
+ обработка медленных или долгоживущих соединений
− тяжело программировать
− использование блокирующий вызоыв все портит

Кто есть кто

- **Apache** – prefork, worker, threads, C
- **Tomcat**, **Jetty** – threads, Java
- **Starman**, **Gunicorn** – prefork, языки высокого уровня
- **Nginx**, **Lighttpd** – асинхронные, C
- **Node.JS**, **Tornado** – асинхронные, языки высокого уровня

Frontend и Backend

Общая архитектура



Задачи Frontend (web) сервера

- отдача статических документов
- проксирование (reverse proxy)
- балансировка нагрузки
- кеширование
- сборка SSI
- авторизация, SSL, нарезка картинок, gzip

Настройка проксирования в nginx

```
proxy_set_header Host      $host;
proxy_set_header X-Real-IP $remote_addr;
location / {
    proxy_pass http://backend;
}
location /partner/ {
    proxy_pass http://www.partner.com;
}
location ~ \.\w\w\w?\w?$ {
    root /www/static;
}
```

Настройка upstream в nginx

```
upstream backend  {
    server back1.example.com:8080 weight=1 max_fails=3;
    server back2.example.com:8080 weight=2;
    server unix:/tmp/backend.sock;
    server backup1.example.com:8080 backup;
    server backup2.example.com:8080 backup;
}
```

Backend (application) сервер

Роль application сервера заключается в исполнении бизнес логики приложения и генерации динамических документов.

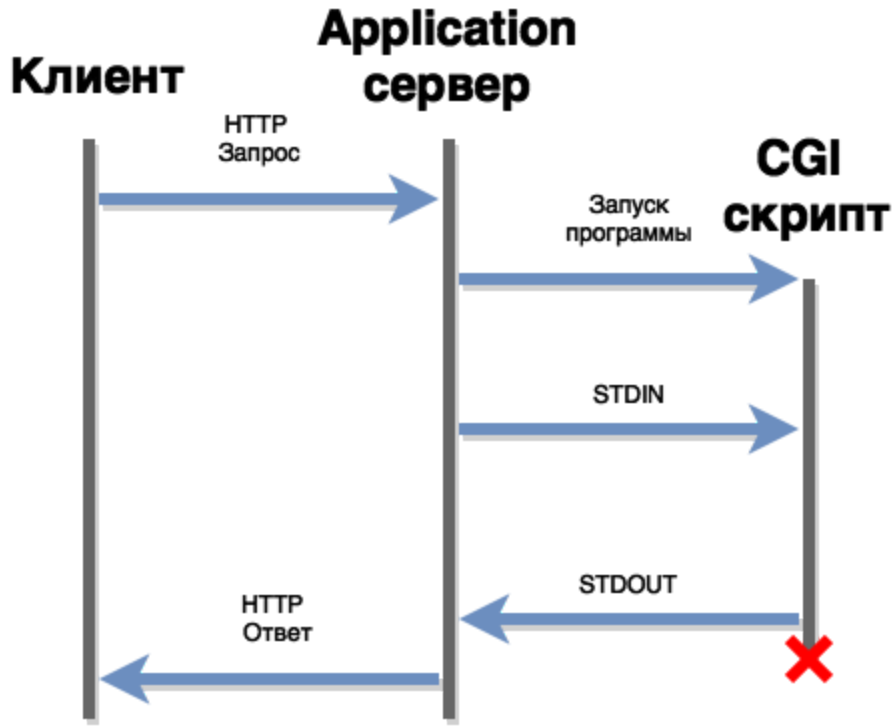
На каждый HTTP запрос application сервер запускает некоторый обработчик в приложении. Это может быть функция, класс или программа, в зависимости от технологии.

Протоколы запуска приложения

- Servlets и др. специализированное API
- mod_perl, mod_python, mod_php
- CGI
- FastCGI
- SCGI
- PSGI, **WSGI**, Rack

CGI - Common Gateway Interface

- Метод, QueryString, заголовки запроса - через **переменные окружения**
- Тело запроса передается через **STDIN**
- Заголовки и тело ответа возвращаются через **STDOUT**
- HTTP код ответа передается через псевдозаголовок **Status**
- Поток ошибок **STDERR** направляется в лог ошибок сервера

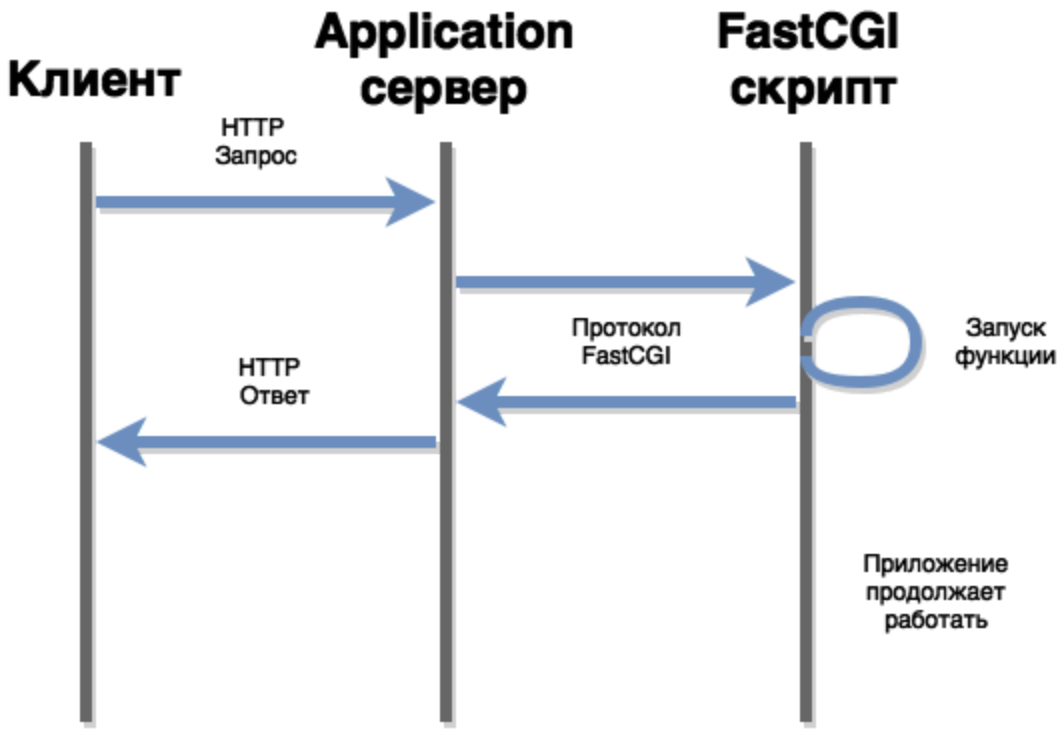

Клиент
Application сервер
CGI скрипт
HTTP Запрос
Запуск программы
STDIN
STDOUT
HTTP Ответ

Переменные окружения CGI

- `REQUEST_METHOD` - метод запроса
- `PATH_INFO` - путь из URL
- `QUERY_STRING` - фрагмент URL после `?`
- `REMOTE_ADDR` - IP адрес пользователя
- `CONTENT_LENGTH` - длина тела запроса
- `HTTP_COOKIE` - Заголовок `Cookie`
- `HTTP_ANY_HEADER_NAME` - любой другой HTTP заголовок

FastCGI и SCGI

Основная проблема CGI - низкая производительность. Протоколы FastCGI и SCGI призваны решить эту проблему путем демонизации приложения. Иногда это возможно сделать даже без изменения кода CGI приложения.


Клиент
Application сервер
FastCGI скрипт
HTTP Запрос
Протокол FastCGI
Запуск функции
HTTP Ответ
Приложение продолжает работать

WSGI - актуальный протокол

WSGI, PSGI, Rack - протоколы вызова функции обработчика из application сервера. Сам application server при этом может выполняться в отдельном процессе или совпадать с web сервером. Как правило, при использовании этих протоколов в качестве application сервера выступает отдельный легковесный процесс.

WSGI - обработчик

```
TODO: from here
```