

Web сервера



Запуск web сервера

- Команда на запуск

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

ДИРЕКТОРИЯ С РАЗЛИЧНЫМИ КОНФИГАМИ

В СЛУЧАЕ ИЗМЕНЕНИЯ РАБОТЫ ЧИТАЕМ НОВЫЙ КОНФИГ
 - Чтение файла конфигурации
 - Получение порта 80 + 443
 - Открытие (создание) логов
 - Понижение привилегий
 - Запуск дочерних процессов/потоков (*)
 - Готов к обработке запроса
- ДИРЕКТОРИЯ ДЛЯ ЗАПУСКА ДЕМОНА(ПРОГРАММА, РАБОТАЮЩАЯ В ФОНЕ, ПИШУЩАЯ В ПЕРЕОПРЕДЕЛЕННЫЕ STDIN/STDOUT)
- ЗАЩИТА ОТ УЯЗВИМОСТЕЙ ОТ ЗАПРОСОВ, ПРИХОДЯЩИХ НА WEB-СЕРВЕР

Файлы web сервера

- Конфиг `/etc/nginx/nginx.conf`
 - ПИШЕТСЯ, ОТСТРАИВАЕТСЯ, ДАЛЕЕ В РОДИТЕЛЬСКИЙ КОНФИГ ВКЛЮЧАЮТСЯ ДР КОНФИГИ, КОТОРЫЕ ДОЛЖНЫ РАБОТАТЬ, УДОБНО В АДМИНИСТРИРОВАНИИ(RELOAD/RESTART SIMULINK, ССЫЛКА НА SITES-AVAILABLE B SITES-ENABLED
- Init-скрипт `/etc/init.d/nginx`
 - `{start|stop|restart|reload|status}`
 - RESTART - ПОЛНАЯ ПЕРЕЗАГРУЗКА С КРАТКОСРОЧНОЙ ОСТАНОВКОЙ ДЕМОНА, УЯВЗИМОСТЬ ПРИ ВЫСОКОЙ НАГРУЗКЕ
 - RELOAD - МЯГКАЯ ПЕРЕЗАГРУЗКА(ПЕРЕЧИТАТЬ НОВЫЙ КОНФИГ)
- PID-файл `/var/run/nginx.pid`
 - отправка сигналов процессу
- Error-лог `/var/log/nginx/error.log`
- Access-лог `/var/log/nginx/access.log`
 - ИНФОРМАЦИЯ О ЗАПРОСАХ
 - ОДНА СТРОКА - ОДНА ПАРА ЗАПРОС - ОТВЕТ

Процессы web сервера

- Master (root, 1 процесс)
 - Чтение и валидация конфига
 - Открытие сокета (ов) и логов (СОКЕТЫ - ПЕРВОНАЧАЛЬНО ОТ ПРИВЕЛЕГИРОВАННЫХ ПОРТОВ 80/443)
 - Запуск и управление дочерними процессами (worker)
 - Graceful restart, Binary updates
- Worker (www-data, 1+ процессов)
 - Обработка входящих запросов

Читаем HTTP запрос
Выделяем память под
запрос
Выделяем
filehandler(указатель
на файл для чтения
тела запроса по
частям) для обработки
тела запроса



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_lua, mod_perl, mod_gzip

есть вариант слияния исходников и
перекомпиляции бинарного файла(out of date)

.so объект сливается с конфигом

РАСШИРЕНИЯ ПОЗВОЛЯЮТ РАСШИРЯТЬ ФУНКЦИОНАЛ WEB
СЕРВЕРА

РАБОТА С
РЕДИРЕКТАМИ

СЖАТИЕ НА ЭТАПЕ
ПРИМЕНЕНИЯ ФИЛЬТРОВ

АВТО ОПРЕДЕЛЕНИЕ ТИПА ДЛЯ MIME-TYPE
ПО РАСШИРЕНИЮ/СИГНАТУРЕ БИНАРНОГО
ФАЙЛА

Конфигурация web сервера

Терминология

virtual host, server - секция конфига web сервера, отвечающая за обслуживание определенной группы доменов

location - секция конфига, отвечающая за обслуживание определенной группы URL

directive {} - контекстные директивы, остальные директивы - в рамках определенного контекста

```
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 $status';
    server {
        listen 80;
        server_name one.example.com www.one.example.com;
        access_log /var/log/nginx.access_log simple;
        location / {
            root /www/one.example.com;
        }
        location ~* ^.+\. (jpg|jpeg|gif)$ {
            root /www/images;
            access_log off;
            expires 30d;
        }
    }
}
```

ФОРМАТ
ЛОГА

ГРУППА ДОМЕНОВ ДЛЯ
ОБРАБОТКИ

МОЖЕТ БЫТЬ REGEX

РЕГЕХ: PCRE

ДИРЕКТИВА, ОТКУДА ОТДАЕМ
ФАЙЛЫ


ДЕФОЛТНЫЙ LOCATION(
НЕ НАЙДЕМ КООКНЕТЫЙ
LOCATION

SERVER - ЗАДАЕТ ГРУППУ ДОМЕНОВ ДЛЯ
ОБРАБОТКИ

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

1. `location = /img/1.jpg` LOCATION ПО ТОЧНОМУ СОВПАДЕНИЮ, НАИВЫСШИЙ ПРИОРИТЕТ
2. `location ^~ /pic/` ПРИОРИТЕТНЫЙ ПРЕФИКСНЫЙ LOCATION (ДЛЯ ОТМЕНЫ ПРОВЕРКИ РЕГУЛЯРОК)
3. `location ~* \.jpg$` REGEX
4. `location /img/` ОБЫЧНЫЙ ПРЕФИКСНЫЙ LOCATION  ЛЮБОЙ ТЕКСТ

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

Примеры: `/img/1.jpg` `/img/2.jpg` `/img/2.png` `/pic/1.jpg`

КАКОЙ-ТО
ЗАПРОС

1



3

12

4

2

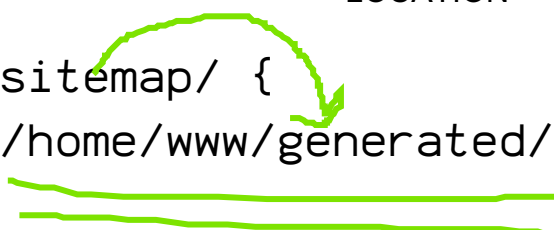
Алгоритм выбора location

1. Ищем полное совпадение по `location = /img/1.jpg`
2. Ищем максимальный префиксный location `location ^~ /pic/`
или `location /img/`
 MAX LEN (НАИБОЛЬШЕЕ СОВПАДЕНИЕ)
3. Если location содержит `^~`, то location найден
4. Проверяем все location с регулярным выражением `location ~*`
 REGEX
`\.jpg$`, отдаем первый совпавший
5. Если ни одно регулярное выражение не подошло, отдаем location без спецификаторов из пункта 2

Отдача статических документов

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

ALIAS - НЕ РАБОТАЕТ С
REGEX



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

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

Атрибуты 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, чтобы открыть файл, необходимо иметь права на чтение `r` самого файла и на исполнение `x` директорий, в которых он находится. Наличие прав проверяется следующим образом:

ПРОВЕРЯЕМ ПРАВА ДИРЕКТОРИЙ

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

Модели
обработки

сетевых
соединений

Простейший ТСР 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()
```

ИМЕННО СЕТЕВОЙ, НЕ UNIX SOCKET

ТСР

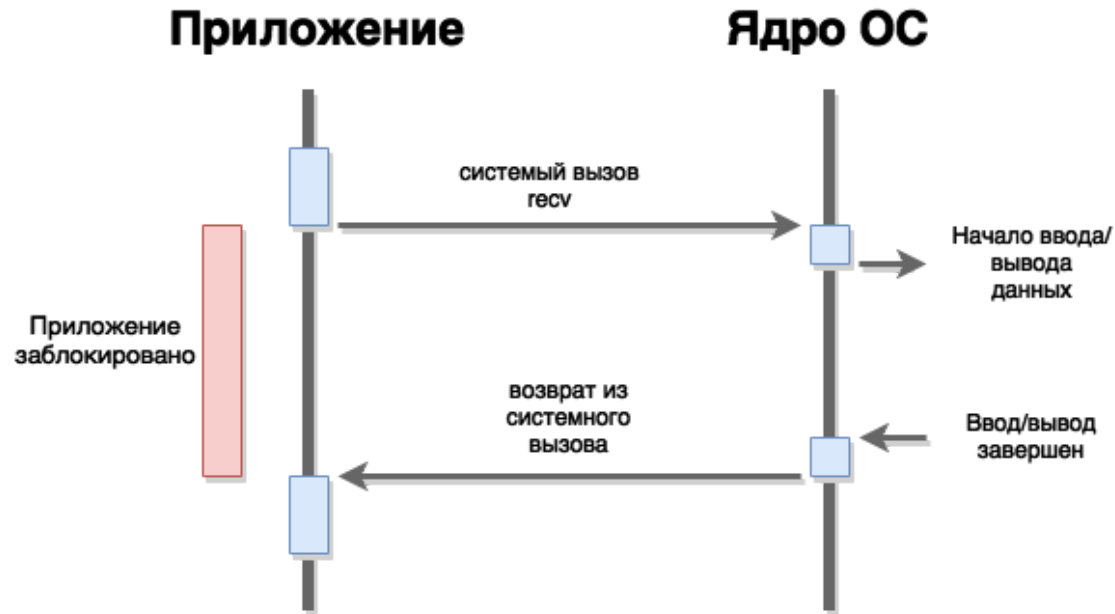
ДЕЛАЕМ FORK НА N ПРОЦЕССОВ, MASTER РЕГУЛИРУЕТ НАГРУЗКУ МЕЖДУ НИМИ

DELIMITERS IN HTTP

location / {
 root
 /www;
}

Блокирующий ввод-вывод

ЖДЕМ, ПОКА `recv()` НАСЧЕТ ЧИТАТЬ ДАННЫЕ С КЛИЕНТСКОГО СОКЕТА НА СЕРВЕРЕ, ИХ МОЖЕТ И НЕ БЫТЬ КАКОЕ-ТО ВРЕМЯ, ИЗ-ЗА ЭТОГО ПРОЦЕСС БЛОКИРУЕТСЯ



Решение проблемы

- множество потоков - multithreading
- множество процессов - prefork, pool of workers СОЗДАНИЕ НЕСКОЛЬКИХ ДОЧЕРНИХ ПРОЦЕССОВ
- комбинированный подход

Плюсы и минусы prefork

- + простота разработки
- + можно использовать любые библиотеки (В Т.Ч. НЕ ПОТОКОБЕЗОПАСНЫЕ)
- большое потребление памяти: 1 клиент = 1 процесс
- проблема с долгоживущими соединениями

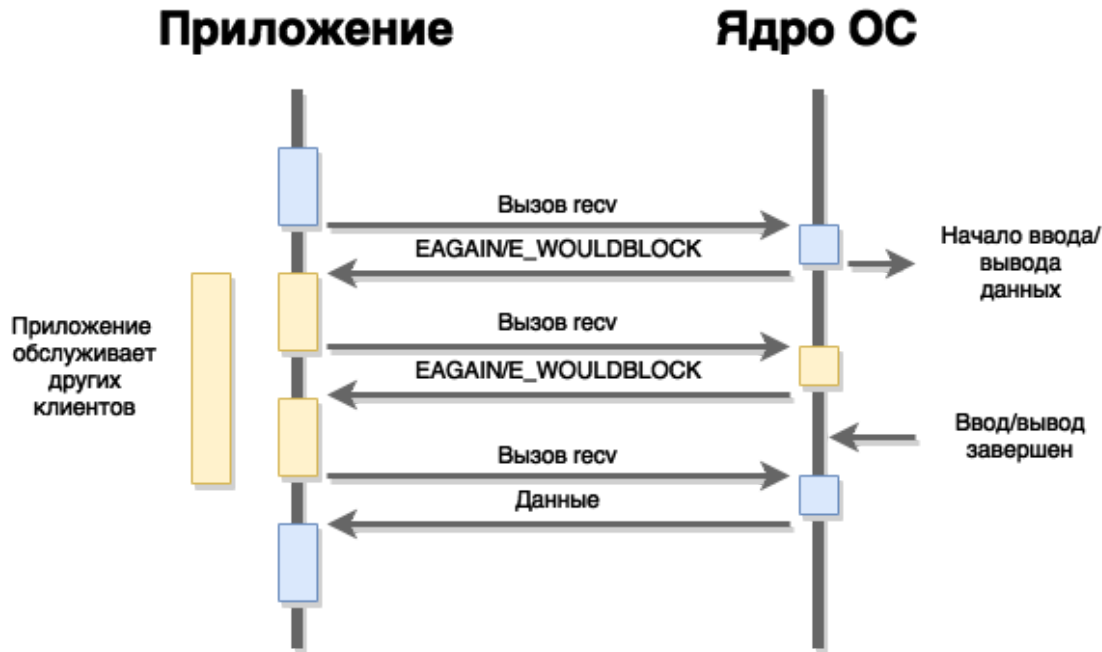
НАПРИМЕР, ЕСЛИ СОЕДИНЕНИЙ ОЧЕНЬ МНОГО, А СКОРОСТЬ СЕТИ МЕДЛЕННАЯ, ТО РАНО ИЛИ ПОЗДНО ВСЕ ВОРКЕР-ПРОЦЕССЫ ЗАБЛОКИРУЮТСЯ НА ОЖИДАНИИ ДАННЫХ ОТ КАЖДОГО ИЗ СОЕДИНЕНИЙ

Плюсы и минусы multithreading

По сравнению с prefork,

- + экономия памяти: 1 клиент = 1 поток ИСПОЛЬЗУЮТ SHARED MEMORY(ОТСЮДА LOCK SEMAPHORE, ЗАЩИТА ОДНОВРЕМЕННОГО ДОСТУПА К ОТДНОЙ ЯЧЕЙКЕ ПАМЯТИ)
- требует аккуратной работы с памятью
- как следствие, накладывает ограничение на выбор библиотек

Неблокирующий ввод-вывод

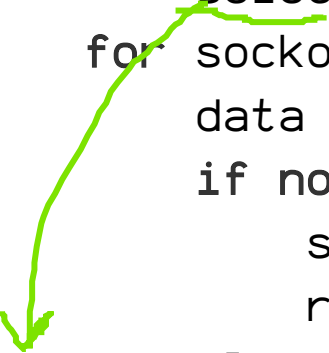


ОБСЛУЖИВАЕМ НЕСКОЛЬКО СОКЕТОВ(СЛЕДИМ ЗА ИХ СОСТОЯНИЕМ) - В СЛУЧАЕ

ПРИХОДА НА ОДИН ИЗ СОКЕТОВ НОВОЙ ИНФОРМАЦИИ(READY TO READ), ОН БУДЕТ ОБРАБОТАН НЕКИМ HANDLER'ОМ/CALLBACK'ОМ, КАК ТОЛЬКО ДО НЕГО ДОЙДЕТ ОЧЕРЕДЬ В ЦИКЛЕ СОБЫТИЙ

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))
```



СИСТЕМНЫЙ
ВЫЗОВ

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 блокирующих вызовов 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