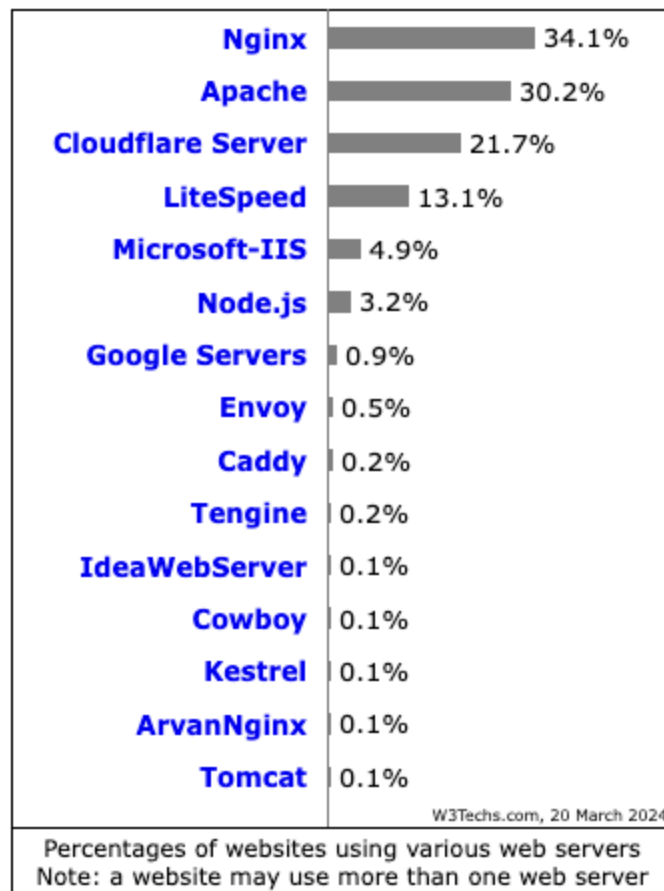


Web сервера



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|reload|status}`
- 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



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

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

```

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;
        }
    }
}

```

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.

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

Алгоритм выбора location

1. Ищем полное совпадение по `location = /img/1.jpg`
2. Ищем максимальный префиксный location `location ^~ /pic/`
или `location /img/`
3. Если location содержит `^~`, то location найден
4. Проверяем все location с регулярным выражением `location ~* \.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;  
}  
location /sitemap/ {  
    alias /home/www/generated/;  
}
```

/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:

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

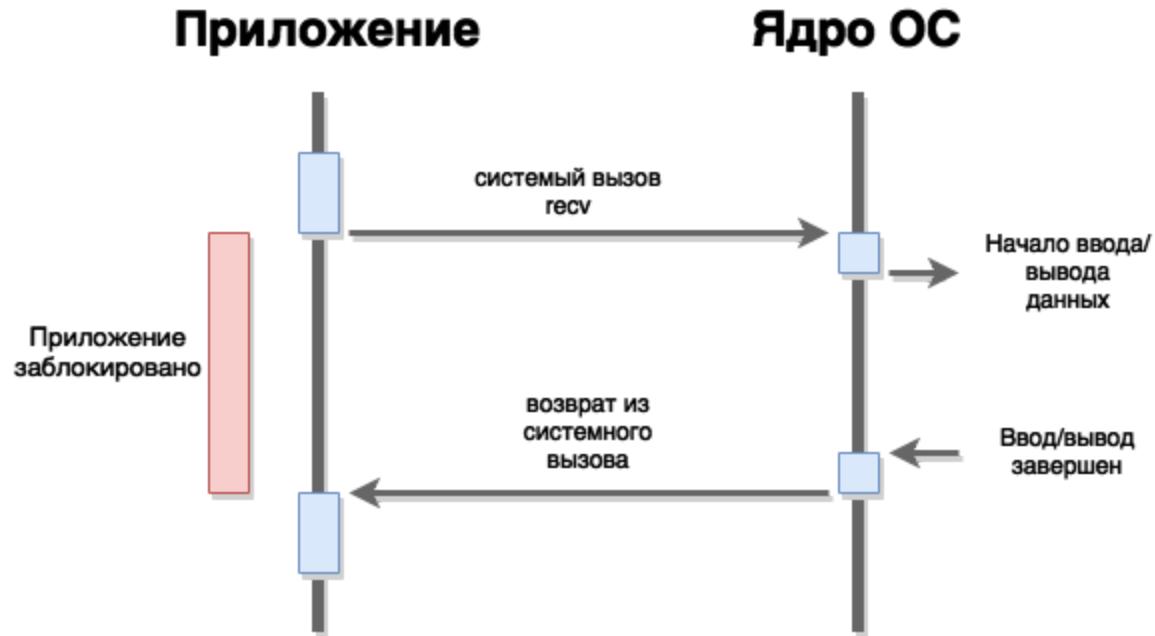
Модели
обработки

сетевых

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()
```

Блокирующий ввод-вывод



Решение проблемы

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

Плюсы и минусы prefork

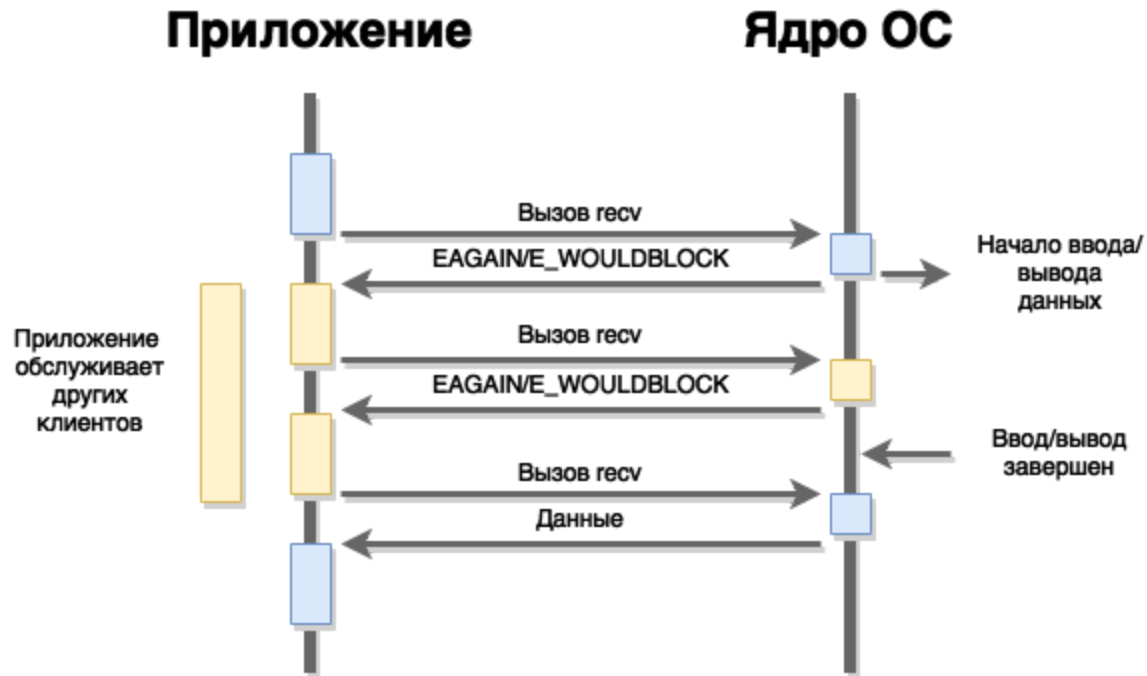
- + простота разработки
- + можно использовать любые библиотеки
- большое потребление памяти: 1 клиент = 1 процесс
- проблема с долгоживущими соединениями

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 блокирующих вызовов все портит

Кто есть кто

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