РОССИЙСКИЕ ИНТЕРНЕТ-ТЕХНОЛОГИИ
ВЫСОКИЕ НАГРУЗКИ

On High-Performance Server Design

Антон Самохвалов

Яндекс

Проблема C10K

- Почему обслужить 10000 одновременных соединений сложно?
- Железо — больше не проблема.
- Тюнинг софта и OS.

Один запрос за другим

```
while (fd = accept(...)) {
    char buf[BUFLEN];
    //прочесть данные от клиента
    int len = read(fd, buf,
 sizeof(buf));
    //обработать их
    const char* out =
 process_data(buf, len);
    //отдать результат
```

Процессы

- программы падают;
- код нужно изолировать;
- пул соединений к MySQL? Keep-Alive?

```
while (fd = accept()) {
    if (fork() == 0) {
        //потомок
        ClientFunc(fd);
    }
```

Потоки

```
while (fd = accept(...)) {
    //получить свободный поток
    Thread* thr = GetThreadFromPool();
    //положить запрос на обработку этим потоком
    thr->Process(fd, ClientFunc);
}
```

Блокирующий IO

- Read()/write()#
- Не просто, а очень просто
  - «чище» код;
  - меньше ошибок.
- В тредах — очень даже ничего
  - несмотря на все слухи, это эффективно.

Select, kqueue, epoll...

- Код в виде state-машины(FSM)#
- Не подходит для диска

```
while (event = poller->Wait()) {
  switch (event->type) {
    case ReadyRead:
      int len = read(event->fd, event->buf
                                , event->blen);
      ProcessInput(event->buf, event->blen);
      poller->Add(event, Read | Write);
  ....
```

Потоки vs. FSM

```
//прочитать данные
char buf[BUF];
int len = read(fd, buf, BUF);
//привести к верхнему регистру
for (int i = 0; i < len; ++i)
    buf[i] = ToUpper(buf[i]);
//отдать результат
write(fd, buf, len);
```

```
switch (event->state) {
  case Read:
    ctx->len = read(fd, ctx->buf,
                    ctx->buflen);
    event->state = Process; break;

  case Process:
    for (int i = 0; i < ctx->len; ++i)
      ctx->buf[i] = ToUpper(ctx->buf[i]);
    poller->Add(event, Write); break;

  case Write:
    write(fd, ctx->buf, ctx->len);
    return;
}
```

- Много кода
- Дуализм!

Co-Routines

- Компромисс между потоками и FSM
  - линейный код;
  - эффективность как и у FSM (setjmp/longjmp);
  - простота реализации.
- 1.000.000 зеленых тредов
  - Yandex.MetaSearch;
  - Yandex.Spider;
  - Yandex.UdpProto.

AIO

- Posix aio_read/aio_write/aio_suspend
- Работает не везде одинаково

- `//попросить систему прочитать еще`
- `ScheduleNextRead(current);`
- `//дождаться результатов прошлого запроса`
- `Buffer last = WaitForComplete();`

«Как это делалось в Одессе»

- Apache 1.x, Apache 2.x
- Nginx, Lighttpd
- FastCGI
- Что лучше?
  - Стек, контекст и аллокатор;
  - Keep-Alive;
  - State-машина — это сложно.

А у нас?

- Потоки на нижних уровнях
  - максимум производительности;
  - репликация -> надежность.
- Процессы на верхних
  - максимум изоляции.
- Никаких state-машин
  - корутины.

Практические советы

- TCP_CORK, TCP_NODELAY
- Zero-copy, счетчики ссылок
- Кеширование
  - аллокатор;
  - пулы тредов.
- Платформонезависимость
- Lock contention

Заключение

- Все это ерунда, и мы говорили о 5% производительности
- Apache 2 vs. Lighttpd vs Nginx.
  - только первый является сервером общего назначения.
- Специализация
  - балансировка нагрузки 1000 rps на 100 машин.


That's all Folks!
Cartoon Songs From
MERRIE MELODIES & LOONEY TUNES