Nginx 在windows系统下优化

Nginx Openresty For Windows (NOW)开源项目实践

关于我们

- 360企业安全
- 提供私有云安全管理平台产品
- 服务器每天处理终端的安全策略下发和日志上传等
- 单服务器支撑最大安全终端数 35000 以上
- 超过 10000 企业用户部署了私有云安全服务器

项目简介

- 关于 Nginx
  - 基于异步模型、高性能、web 服务器、开源软件
  - Nginx.org
- 关于 Openresty
  - 核心 ---- nginx_lua 模块
  - 追求更高开发效率
  - Openresty.org
- 关于NOW
  - Nginx Openresty For Windows （NOW）私有云安全服务器的基础组件
  - https://github.com/LomoX-Offical/nginx-openresty-windows
  - 追求在 windows 平台上更高性能

开始立项--不想用 C 写服务器了

- 开始做技术选型
  - Web application ------- apache, nginx
  - Script language --------- php, js, lua, clojure

- 选择了 nginx openresty
  - 看中了 ngx-lua-module 模块
  - Lua-jit 的效率在技术选型范围内的语言中性能最高

Ngx-lua-module 的开发优势

让服务器开发更简单

Example： nginx.conf

```
location /hello {
        content_by_lua_block {
                ngx.say("hello world!");
        }
}
```

Ngx-lua-module 也有用到协程

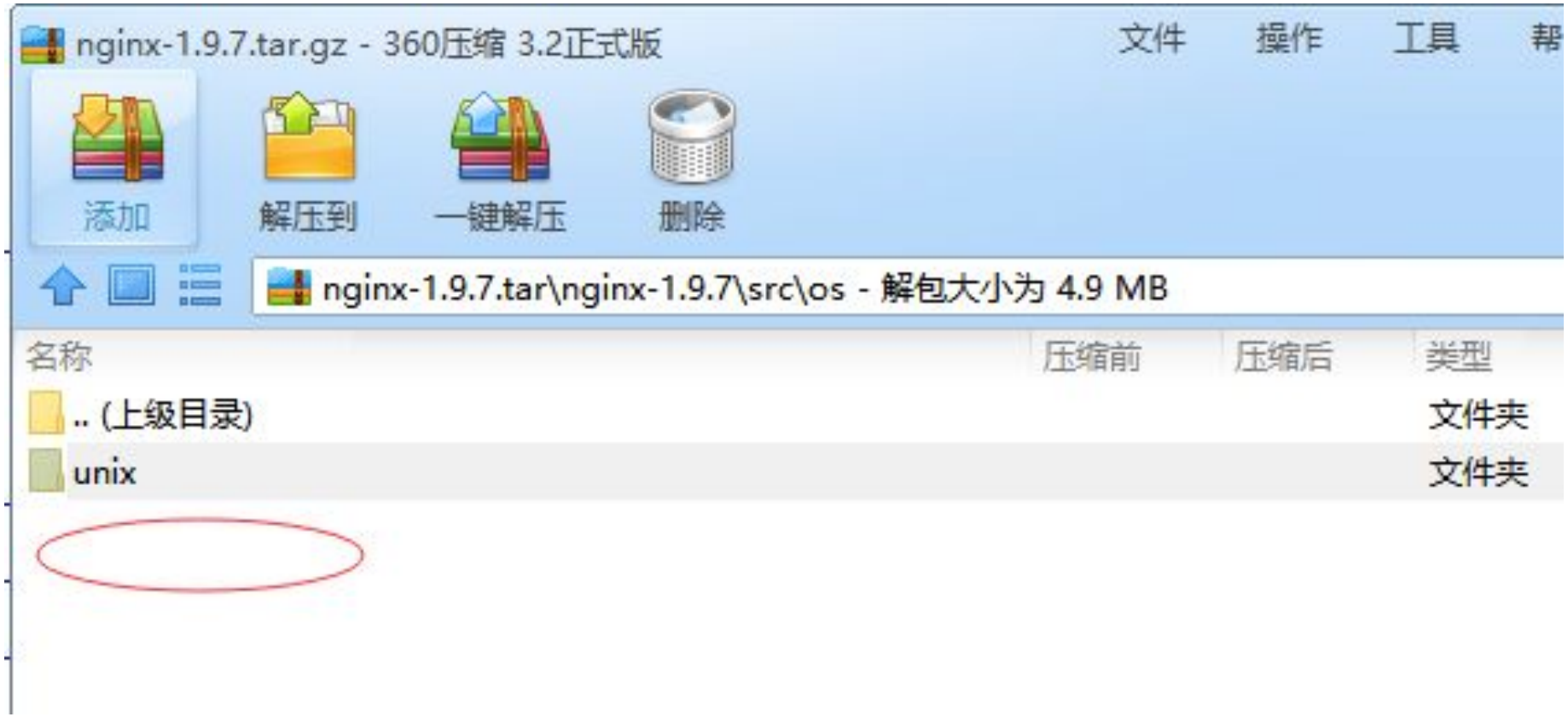
```
location /test {
    resolver 8.8.8.8;

    content_by_lua '
        local sock = ngx.socket.tcp()
        local ok, err = sock:connect("www.google.com", 80)
        if not ok then
            ngx.say("failed to connect to google: ", err)
            return
        end
        ngx.say("successfully connected to google!")
        sock:close()
    ';
}
```

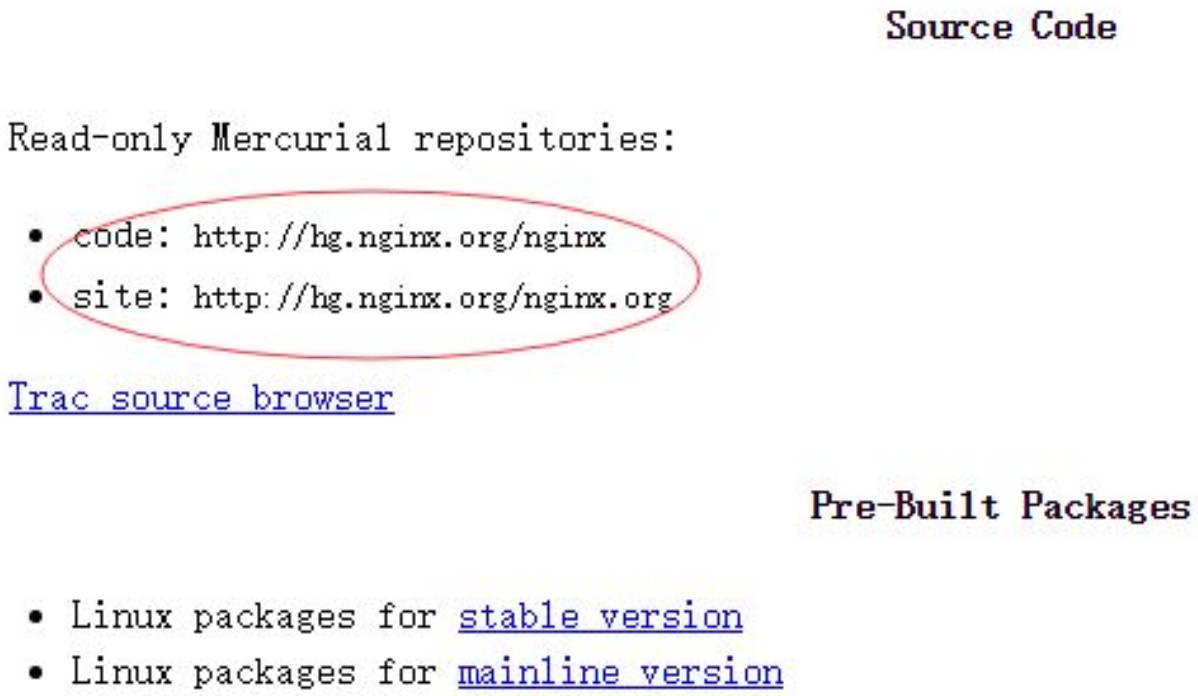
碰到的坑

- ngx-lua-module 和 openresty 没有 windows 版本
- 需要把 nginx 和 openresty 在 windows 上编译
- 不管是 nginx 或者 openresty 官网都没有 windows 系统逻辑代码

此处应有 win32 文件夹



官网页面最下方



Source Code

Read-only Mercurial repositories:

- code: http://hg.nginx.org/nginx
- site: http://hg.nginx.org/nginx.org

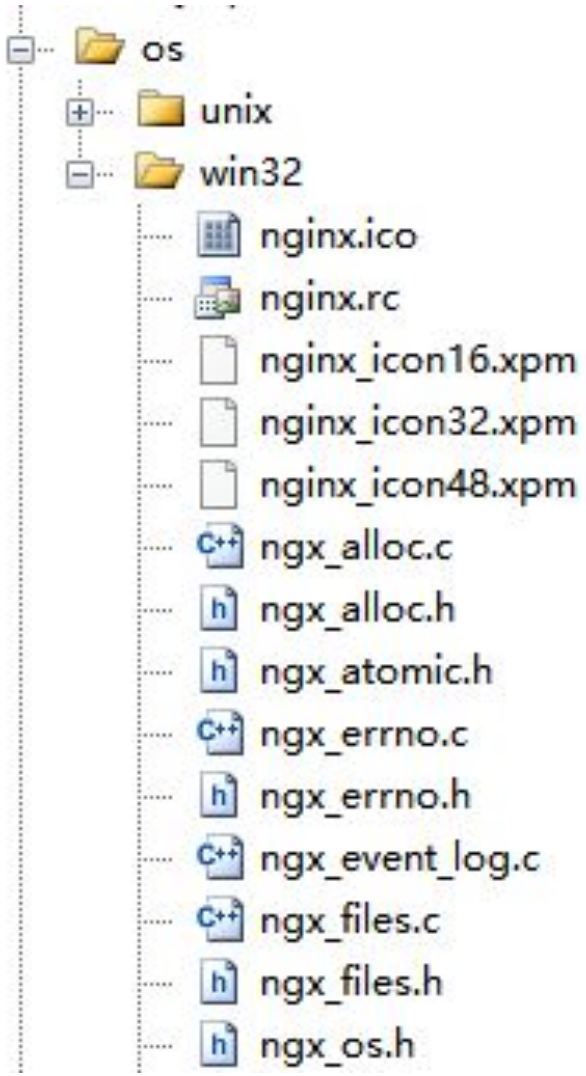
Trac source browser

Pre-Built Packages

- Linux packages for stable version
- Linux packages for mainline version

http://hg.nginx.org/nginx 是一个 HG 版本控制库

终于有代码了


os
unix
win32
nginx.ico
nginx.rc
nginx_icon16.xpm
nginx_icon32.xpm
nginx_icon48.xpm
ngx_alloc.c
ngx_alloc.h
ngx_atomic.h
ngx_errno.c
ngx_errno.h
ngx_event_log.c
ngx_files.c
ngx_files.h
ngx_os.h

编译工作

- 依赖库
  - Zlib
  - PCRE
  - OpenSSL
- 修改 configure、auto 脚本等
- 使用 Visual Studio 的 nmake 进行编译
- 进行单元测试（nginx 官方以及 openresty）

与官网 windows 版本对比

- 官网 windows 版本
  - 单工作进程
  - 1024并发限制
  - 1.9 版本以前，无法使用共享内存

- Now 版本
  - 多工作进程
  - 超过 32768 并发连接
  - 可以使用共享内存

子进程启动方式（主要的差异）

- Windows
  - CreateProcess
  - 子进程从 main 函数从头开始，资源以及内存默认都没有继承下来

- Linux
  - fork
  - 子进程从 fork 函数之后开始，所有资源内存布局均与父进程一样

共享内存

- Windows
  - CreateFileMapping
  - MapViewOfFileEx
  - 根据映射名字获得内存地址

- Linux
  - mmap
  - 得益于 fork，父进程一次分配，子进程均可共享

共享内存坑一（无法使用共享内存）

- 在使用 limit_conn 系列指令的时候报错：
  - limit_conn_zone $binary_remote_addr zone=addr:10m;
  - shared zone "addr" has no equal addresses: 02A30000 vs 031B0000

- Nginx 使用共享内存的方式是 slab -> rbtree

- 子进程共享内存地址不一致

- MapViewOfFileEx 在 MSDN 上描述，同样的命名，同样的大小，无法保证所有进程获取同样的地址。

共享内存坑一解决方案

- 设定初始地址
  - #define NGX_SHMEM_BASE 0x2efe0000
  - #define NGX_SHMEM_BASE 0x0000047047e00000 (64bit)
  - static u_char * base = NGX_SHMEM_BASE
- 调用 MapViewOfFileEx 函数时，使用 base 指针地址
- 下一块共享内存的 base 指针，带一些对齐信息
  - base += ngx_align(size, ngx_allocation_granularity);

- 在共享内存申请顺序与大小相同的情况下，可以保证指针一致
- nginx 1.9 之后官方代码实现，我们在1.2已经使用上

共享内存坑一解决方案

```
#ifdef _WIN64
#define NGX_SHMEM_BASE  0x0000020000000000
#else
#define NGX_SHMEM_BASE  0x20000000
#endif

static u_char  *base = (u_char *) NGX_SHMEM_BASE;
shm->handle = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE,
    (u_long) (size >> 32),
    (u_long) (size & 0xffffffff),
    (char *) name);

shm->addr = MapViewOfFileEx(shm->handle, FILE_MAP_WRITE, 0, 0, 0, base);
if (shm->addr != NULL) {
    base += ngx_align(size, ngx_allocation_granularity);
    return NGX_OK;
}
```

共享内存坑二

- Nginx 使用 slab 但没有初始化slab参数，默认为0:
  - ngx_slab_max_size
  - ngx_slab_exact_size
  - ngx_slab_exact_shift

- 永远使用完整的一个 Page 去存储 小于 1 Page 大小的数据。
  - ngx_slab_alloc_locked 实现使用上了 ngx_slab_max_size

- 这是由于 windows 的启动流程导致的逻辑差异

共享内存坑二解决方案

- 调整 nginx 在 windows 上的代码结构
- 把 ngx_slab_max_size 等变量的初始化提前到共享内存分配之前
  - 新增一个函数 ngx_slab_module_init
- 官方暂未解决

处理性能优化——单进程限制

- 官网 windows 版本
  - 主进程启动侦听端口 listen ，然后立即关闭端口 closesocket
  - 子进程各自打开侦听端口
  - 只有第一个子进程能 accept 到新连接

- NOW版本
  - 主进带着环境变量启动的子进程，根据环境变量获得 fd
  - 程启动侦听端口 listen，把 fd 写入环境变量
  - 所有的子进程，都可以 accept 到新连接

网络性能优化—— 1024 连接数限制

- Winsock2.h 里边的定义

```
#define FD_SETSIZE  64

typedef struct fd_set {
    u_int    fd_count;               /* how many are SET? */
    SOCKET   fd_array [FD_SETSIZE];  /* an array of SOCKETs */
} fd_set;
```

- 官网 windows 版本

```
#define FD_SETSIZE  1024
```

- NOW 版本

```
#define FD_SETSIZE  32768
```

网络模型的思考

- 官网版本使用 select 模型
- Select 有FD_SETSIZE 的大小限制，系统默认 64
- 官网的select 模型复杂度为 O(n*n)
- 更好的实现 libevent 使用哈希可以降到 O(n)

官网 select 模型

```
static ngx_event_t  **event_index;

work_read_fd_set = master_read_fd_set;
work_write_fd_set = master_write_fd_set;
ready = select(0, &work_read_fd_set, &work_write_fd_set, NULL, tp);
for (i = 0; i < nevents; i++) {
    ev = event_index[i];
    c = ev->data;
    found = 0;
    if (ev->write) {
        if (FD_ISSET(c->fd, &work_write_fd_set)) {
            found = 1;
        }
    } else {
        if (FD_ISSET(c->fd, &work_read_fd_set)) {
            found = 1;
        }
    }
    if (found) {
        ev->ready = 1;
        ngx_post_event(ev, ngx_posted_events);
        nready++;
    }
}
```

Libevent 的select模型

```
res = select(fd_count,
        (struct fd_set*)win32op->readset_out,
        (struct fd_set*)win32op->writeset_out,
        (struct fd_set*)win32op->exset_out, tv);

if (win32op->readset_out->fd_count) {
    i = rand() % win32op->readset_out->fd_count;
    for (j=0; j<win32op->readset_out->fd_count; ++j) {
        if (++i >= win32op->readset_out->fd_count)
            i = 0;
        s = win32op->readset_out->fd_array[i];
        evmap_io_active(base, s, EV_READ);
    }
}
if (win32op->writeset_out->fd_count) {
    SOCKET s;
    i = rand() % win32op->writeset_out->fd_count;
    for (j=0; j<win32op->writeset_out->fd_count; ++j) {
        if (++i >= win32op->writeset_out->fd_count)
            i = 0;
        s = win32op->writeset_out->fd_array[i];
        evmap_io_active(base, s, EV_WRITE);
    }
}
```

select 设计非常糟糕

大名鼎鼎的 IOCP模型

- 时间复杂度最优 O(1)
- Proactor 模型与 Nginx 本身支持的 Reactor 模型不完全兼容
- MSDN的描述不能支持多进程
- 在多线程情况下，很多 nginx 的模块线程安全无法保障
- 我们经常使用的 nginx-lua-module 他就能保证线程安全

尝试使用 WSAPoll 模型

- 参考 posix 标准的 poll api的功能去实现
- 较容易控制应用层的时间复杂度 O(n)
- 没有 FD_SETSIZE 连接数限制
- Windows Vista 以及 windows 2008 支持

在 Nginx 上使用 WSAPoll 模型

```
typedef struct pollfd {

    SOCKET  fd;
    SHORT   events;
    SHORT   revents;

} WSAPOLLFD, *PWSAPOLLFD, FAR *LPWSAPOLLFD;

static struct pollfd     *event_list;
static ngx_connection_t **connection_list;
static ngx_uint_t         nevents;
```

在 Nginx 上使用 WSAPoll 模型

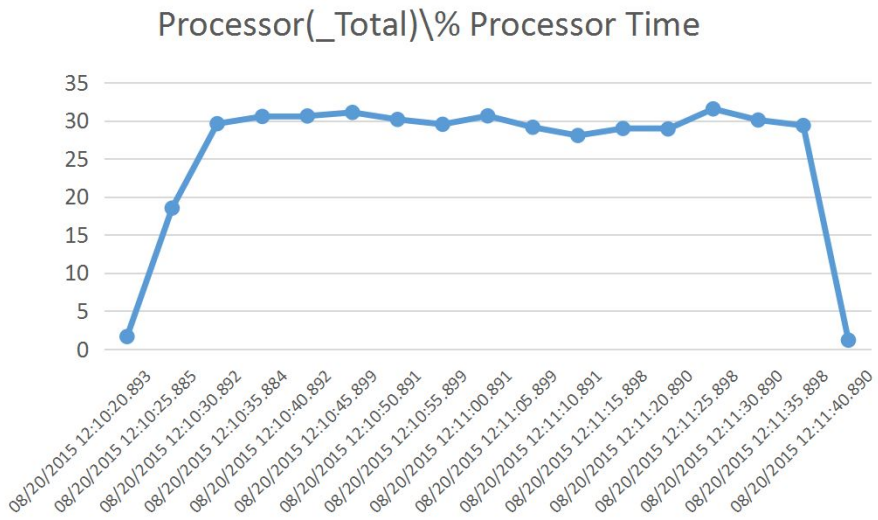
```
ready = ngx_wsapoll(event_list, (u_int) nevents, (int) timer);
for (i = 0; i < nevents && ready; i++) {
    revents = event_list[i].revents;
    if (event_list[i].fd == -1) {
        continue;
    }
    c = connection_list[i];
    found = 0;
    if ((revents & POLLIN) && c->read->active) {
        found = 1;
        ngx_post_event(ev, ngx_posted_events);
    }
    if ((revents & POLLOUT) && c->write->active) {
        found = 1;
        ngx_post_event(ev, &ngx_posted_events);
    }
    if (found) {
        ready--;
        continue;
    }
}
```

性能对比（测试环境）

- OS： Windows Server 2008 R2 Standard X64
- RAM： 4GB
- CPU： Intel Core i5-3570 @3.40GHZ 物理4核心
- 测试参数： ab –kc 200 –n 5000000 172.22.131.45/connect
- /connect URI返回字符串“true”给请求终端

性能对比（官网Nginx）

- CPU

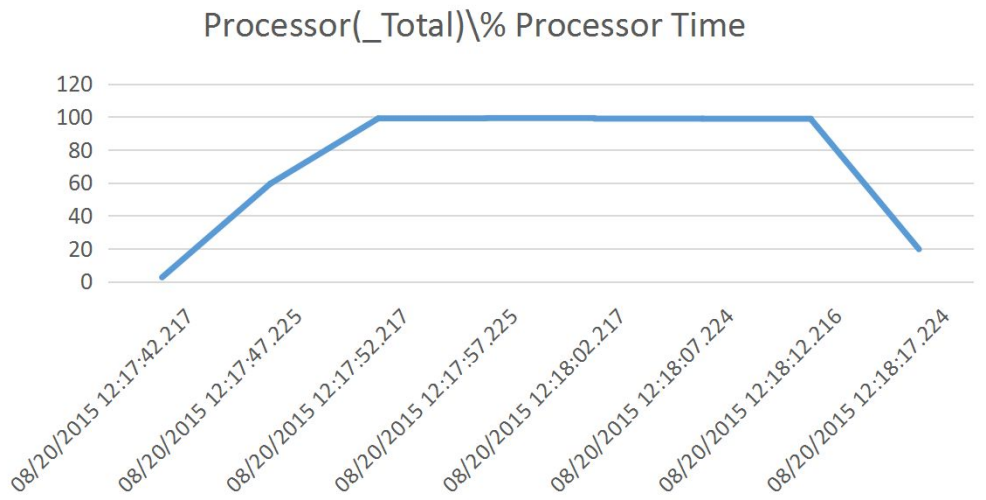


- RPS

```
Concurrency Level:      200
Time taken for tests:   70.718 seconds
Complete requests:      5000000
Failed requests:        0
Write errors:           0
Keep-Alive requests:    4950056
Total transferred:      744750429 bytes
HTML transferred:       20000004 bytes
Requests per second:    70703.33 [#/sec] (mean)
Time per request:       2.829 [ms] (mean)
Time per request:       0.014 [ms] (mean, across all concurrent requests)
Transfer rate:          10284.44 [Kbytes/sec] received
```

性能对比（NOW 32BIT）

- CPU

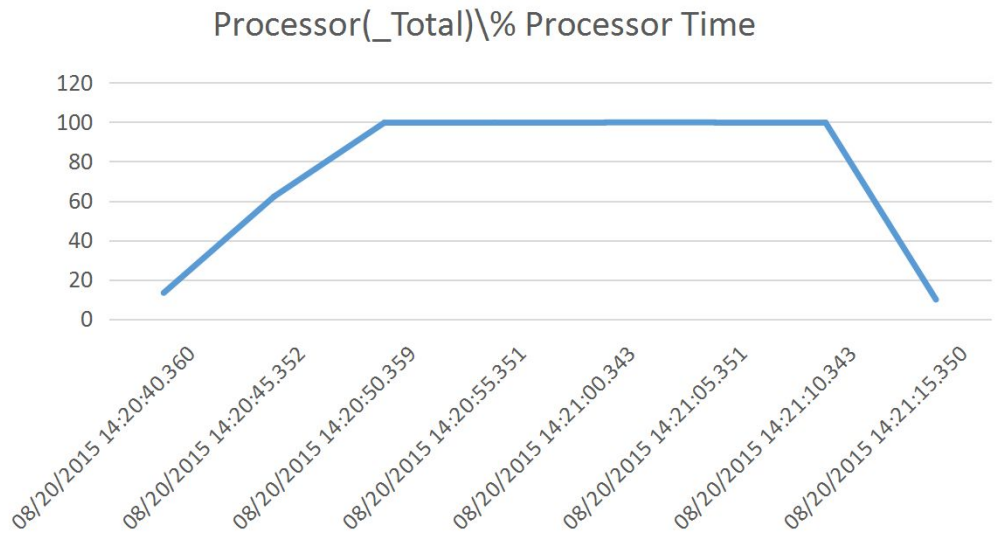


- RPS

```
Concurrency Level:      200
Time taken for tests:   26.117 seconds
Complete requests:      5000000
Failed requests:        0
Write errors:           0
Keep-Alive requests:    4950101
Total transferred:      749759500 bytes
HTML transferred:       25000300 bytes
Requests per second:    191446.02 [#/sec] (mean)
Time per request:       1.045 [ms] (mean)
Time per request:       0.005 [ms] (mean, across all concurrent requests)
Transfer rate:          28034.86 [Kbytes/sec] received
```

性能对比（NOW 64BIT）

- CPU



- RPS

```
Concurrency Level:      200
Time taken for tests:   26.253 seconds
Complete requests:      5000000
Failed requests:        0
Write errors:           0
Keep-Alive requests:    4950091
Total transferred:      749753455 bytes
HTML transferred:       25000100 bytes
Requests per second:    190452.06 [#/sec] (mean)
Time per request:       1.050 [ms] (mean)
Time per request:       0.005 [ms] (mean, across all concurrent requests)
Transfer rate:          27889.08 [Kbytes/sec] received
```

下一步的尝试

- fork 的 windows 实现
- Nginx 1.9 多线程特性的 windows 实现
- IOCP 的多进程使用

谢谢大家

https://github.com/LomoX-Offical/nginx-openresty-windows