src/ - weighttp

- client.c
- client.h
- weighttp.c
- weighttp.h
- worker.c
- worker.h

src/client.c - weighttp

Functions defined

- client_connect
- client_free
- client_io_cb
- client_new
- client_parse
- client_reset
- client_set_events
- client_state_machine

Source code

```
/*
* weighttp - a lightweight and simple webserver benchmarking tool
*
* Author:
*     Copyright (c) 2009-2011 Thomas Porzelt
*
* License:
*     MIT, see COPYING file
*/

 #include "weighttp.h"

 static uint8_t client_parse(Client *client, int size);
 static void client_io_cb(struct ev_loop *loop, ev_io *w, int revents);
 static void client_set_events(Client *client, int events);
 /*
 static void client_add_events(Client *client, int events);
 static void client_rem_events(Client *client, int events);

 static void client_add_events(Client *client, int events) {
     struct ev_loop *loop = client->worker->loop;
     ev_io *watcher = &client->sock_watcher;

     if ((watcher->events & events) == events)
         return;

     ev_io_stop(loop, watcher);
     ev_io_set(watcher, watcher->fd, watcher->events | events);
     ev_io_start(loop, watcher);
 }

 static void client_rem_events(Client *client, int events) {
     struct ev_loop *loop = client->worker->loop;
     ev_io *watcher = &client->sock_watcher;

     if (0 == (watcher->events & events))
         return;

     ev_io_stop(loop, watcher);
     ev_io_set(watcher, watcher->fd, watcher->events & ~events);
     ev_io_start(loop, watcher);
 }
 */

 static void client_set_events(Client *client, int events) {
     struct ev_loop *loop = client->worker->loop;
     ev_io *watcher = &client->sock_watcher;
```

```c

    if (events == (watcher->events & (EV_READ | EV_WRITE)))
        return;

    ev_io_stop(loop, watcher);
    ev_io_set(watcher, watcher->fd, (watcher->events & ~(EV_READ | EV_WRITE)) | events);
    ev_io_start(loop, watcher);
}

Client *client_new(Worker *worker) {
    Client *client;

    client = W_MALLOC(Client, 1);
    client->state = CLIENT_START;
    client->worker = worker;
    client->sock_watcher.fd = -1;
    client->sock_watcher.data = client;
    client->content_length = -1;
    client->buffer_offset = 0;
    client->request_offset = 0;
    client->keepalive = client->worker->config->keep_alive;
    client->chunked = 0;
    client->chunk_size = -1;
    client->chunk_received = 0;

    return client;
}

void client_free(Client *client) {
    if (client->sock_watcher.fd != -1) {
        ev_io_stop(client->worker->loop, &client->sock_watcher);
        shutdown(client->sock_watcher.fd, SHUT_WR);
        close(client->sock_watcher.fd);
    }

    free(client);
}

static void client_reset(Client *client) {
    //printf("keep alive: %d\n", client->keepalive);
    if (!client->keepalive) {
        if (client->sock_watcher.fd != -1) {
            ev_io_stop(client->worker->loop, &client->sock_watcher);
            shutdown(client->sock_watcher.fd, SHUT_WR);
            close(client->sock_watcher.fd);
            client->sock_watcher.fd = -1;
        }

        client->state = CLIENT_START;
    } else {
        client_set_events(client, EV_WRITE);
        client->state = CLIENT_WRITING;
        client->worker->stats.req_started++;
    }

    client->parser_state = PARSER_START;
    client->buffer_offset = 0;
    client->parser_offset = 0;
    client->request_offset = 0;
    client->ts_start = 0;
    client->ts_end = 0;
    client->status_success = 0;
    client->success = 0;
    client->content_length = -1;
    client->bytes_received = 0;
    client->header_size = 0;
    client->keepalive = client->worker->config->keep_alive;
    client->chunked = 0;
    client->chunk_size = -1;
    client->chunk_received = 0;
}

static uint8_t client_connect(Client *client) {
    //printf("connecting...\n");
    start:

```

```c
    if (-1 == connect(client->sock_watcher.fd, client->worker->config->saddr->ai_addr, client->worker-
>config->saddr->ai_addrlen)) {
        switch (errno) {
            case EINPROGRESS:
            case EALREADY:
                /* async connect now in progress */
                client->state = CLIENT_CONNECTING;
                return 1;
            case EISCONN:
                break;
            case EINTR:
                goto start;
            default:
            {
                strerror_r(errno, client->buffer, sizeof(client->buffer));
                W_ERROR("connect() failed: %s (%d)", client->buffer, errno);
                return 0;
            }
        }
    }

    /* successfully connected */
    client->state = CLIENT_WRITING;
    return 1;
}

static void client_io_cb(struct ev_loop *loop, ev_io *w, int revents) {
    Client *client = w->data;

    UNUSED(loop);
    UNUSED(revents);

    client_state_machine(client);
}

void client_state_machine(Client *client) {
    int r;
    Config *config = client->worker->config;

    start:
    //printf("state: %d\n", client->state);
    switch (client->state) {
        case CLIENT_START:
            client->worker->stats.req_started++;

            do {
                r = socket(config->saddr->ai_family, config->saddr->ai_socktype, config->saddr->ai_protocol);
            } while (-1 == r && errno == EINTR);

            if (-1 == r) {
                client->state = CLIENT_ERROR;
                strerror_r(errno, client->buffer, sizeof(client->buffer));
                W_ERROR("socket() failed: %s (%d)", client->buffer, errno);
                goto start;
            }

            /* set non-blocking */
            fcntl(r, F_SETFL, O_NONBLOCK | O_RDWR);

            ev_init(&client->sock_watcher, client_io_cb);
            ev_io_set(&client->sock_watcher, r, EV_WRITE);
            ev_io_start(client->worker->loop, &client->sock_watcher);

            if (!client_connect(client)) {
                client->state = CLIENT_ERROR;
                goto start;
            } else {
                client_set_events(client, EV_WRITE);
                return;
            }
        case CLIENT_CONNECTING:
            if (!client_connect(client)) {
                client->state = CLIENT_ERROR;
                goto start;
            }
        case CLIENT_WRITING:
            while (1) {
                r = write(client->sock_watcher.fd, &config->request[client->request_offset], config->request_size - client->request_offset);
                //printf("write(%d - %d = %d): %d\n", config->request_size, client->request_offset, config->request_size - client->request_offset, r);
                if (r == -1) {
                    /* error */
                    if (errno == EINTR)
                        continue;
                    strerror_r(errno, client->buffer, sizeof(client->buffer));
                    W_ERROR("write() failed: %s (%d)", client->buffer, errno);
                    client->state = CLIENT_ERROR;
                    goto start;
                } else if (r != 0) {
                    /* success */
                    client->request_offset += r;
                    if (client->request_offset == config->request_size) {
                        /* whole request was sent, start reading */
                        client->state = CLIENT_READING;
                        client_set_events(client, EV_READ);
                    }

                    return;
                } else {
                    /* disconnect */
                    client->state = CLIENT_END;
                    goto start;
                }
            }
        case CLIENT_READING:
            while (1) {
                r = read(client->sock_watcher.fd, &client->buffer[client->buffer_offset], sizeof(client->buffer) - client->buffer_offset - 1);
                //printf("read(): %d, offset was: %d\n", r, client->buffer_offset);
                if (r == -1) {
                    /* error */
                    if (errno == EINTR)
                        continue;
                    strerror_r(errno, client->buffer, sizeof(client->buffer));
                    W_ERROR("read() failed: %s (%d)", client->buffer, errno);
                    client->state = CLIENT_ERROR;
                } else if (r != 0) {
                    /* success */
                    client->bytes_received += r;
                    client->buffer_offset += r;
                    client->worker->stats.bytes_total += r;

                    if (client->buffer_offset >= sizeof(client->buffer)) {
                        /* too big response header */
                        client->state = CLIENT_ERROR;
                        break;
                    }
                    client->buffer[client->buffer_offset] = '\0';
                    //printf("buffer:\n==========\n%s\n==========\n", client->buffer);
                    if (!client_parse(client, r)) {
                        client->state = CLIENT_ERROR;
                        //printf("parser failed\n");
                        break;
                    } else {
                        if (client->state == CLIENT_END)
                            goto start;
                        else
                            return;
                    }
                } else {
                    /* disconnect */
                    if (client->parser_state == PARSER_BODY && !client->keepalive && client->status_success
                        && !client->chunked && client->content_length == -1) {
                        client->success = 1;
                        client->state = CLIENT_END;
                    } else {
                        client->state = CLIENT_ERROR;
                    }

                    goto start;
                }
```

```c
            }

        case CLIENT_ERROR:
            //printf("client error\n");
            client->worker->stats.req_error++;
            client->keepalive = 0;
            client->success = 0;
            client->state = CLIENT_END;
        case CLIENT_END:
            /* update worker stats */
            client->worker->stats.req_done++;

            if (client->success) {
                client->worker->stats.req_success++;
                client->worker->stats.bytes_body += client->bytes_received - client->header_size;
            } else {
                client->worker->stats.req_failed++;
            }

            /* print progress every 10% done */
            if (client->worker->id == 1 && client->worker->stats.req_done % client->worker->progress_interval
== 0) {
                printf("progress: %3d%% done\n",
                    (int) (client->worker->stats.req_done * 100 / client->worker->stats.req_todo)
                );
            }

            if (client->worker->stats.req_started == client->worker->stats.req_todo) {
                /* this worker has started all requests */
                client->keepalive = 0;
                client_reset(client);

                if (client->worker->stats.req_done == client->worker->stats.req_todo) {
                    /* this worker has finished all requests */
                    ev_unref(client->worker->loop);
                }
            } else {
                client_reset(client);
                goto start;
            }
    }
}


static uint8_t client_parse(Client *client, int size) {
    char *end, *str;
    uint16_t status_code;

    switch (client->parser_state) {
        case PARSER_START:
            //printf("parse (START):\n%s\n", &client->buffer[client->parser_offset]);
            /* look for HTTP/1.1 200 OK */
            if (client->buffer_offset < sizeof("HTTP/1.1 200\r\n"))
                return 1;

            if (strncmp(client->buffer, "HTTP/1.1 ", sizeof("HTTP/1.1 ")-1) != 0)
                return 0;

            // now the status code
            status_code = 0;
            str = client->buffer + sizeof("HTTP/1.1 ")-1;
            for (end = str + 3; str != end; str++) {
                if (*str < '0' || *str > '9')
                    return 0;

                status_code *= 10;
                status_code += *str - '0';
            }

            if (status_code >= 200 && status_code < 300) {
                client->worker->stats.req_2xx++;
                client->status_success = 1;
            } else if (status_code < 400) {
                client->worker->stats.req_3xx++;
                client->status_success = 1;
            } else if (status_code < 500) {
                client->worker->stats.req_4xx++;
            } else if (status_code < 600) {
                client->worker->stats.req_5xx++;
            } else {
                // invalid status code
                return 0;
            }

            // look for next \r\n
            end = strchr(end, '\r');
            if (!end || *(end+1) != '\n')
                return 0;

            client->parser_offset = end + 2 - client->buffer;
            client->parser_state = PARSER_HEADER;
        case PARSER_HEADER:
            //printf("parse (HEADER)\n");
            /* look for Content-Length and Connection header */
            while (NULL != (end = strchr(&client->buffer[client->parser_offset], '\r'))) {
                if (*(end+1) != '\n')
                    return 0;

                if (end == &client->buffer[client->parser_offset]) {
                    /* body reached */
                    client->parser_state = PARSER_BODY;
                    client->header_size = end + 2 - client->buffer;
                    //printf("body reached\n");

                    return client_parse(client, size - client->header_size);
                }

                *end = '\0';
                str = &client->buffer[client->parser_offset];
                //printf("checking header: '%s'\n", str);

                if (strncmp(str, "Content-Length: ", sizeof("Content-Length: ")-1) == 0) {
                    /* content length header */
                    client->content_length = str_to_uint64(str + sizeof("Content-Length: ") - 1);
                } else if (strncmp(str, "Connection: ", sizeof("Connection: ")-1) == 0) {
                    /* connection header */
                    str += sizeof("Connection: ") - 1;

                    if (strncmp(str, "close", sizeof("close")-1) == 0)
                        client->keepalive = 0;
                    else if (strncmp(str, "Keep-Alive", sizeof("Keep-Alive")-1) == 0)
                        client->keepalive = client->worker->config->keep_alive;
                    else if (strncmp(str, "keep-alive", sizeof("keep-alive")-1) == 0)
                        client->keepalive = client->worker->config->keep_alive;
                    else
                        return 0;
                } else if (strncmp(str, "Transfer-Encoding: ", sizeof("Transfer-Encoding: ")-1) == 0) {
                    /* transfer encoding header */
                    str += sizeof("Transfer-Encoding: ") - 1;

                    if (strncmp(str, "chunked", sizeof("chunked")-1) == 0)
                        client->chunked = 1;
                    else
                        return 0;
                }


                if (*(end+2) == '\r' && *(end+3) == '\n') {
                    /* body reached */
                    client->parser_state = PARSER_BODY;
                    client->header_size = end + 4 - client->buffer;
                    client->parser_offset = client->header_size;
                    //printf("body reached\n");

                    return client_parse(client, size - client->header_size);
                }

                client->parser_offset = end - client->buffer + 2;
            }

            return 1;
        case PARSER_BODY:
            //printf("parse (BODY)\n");
            /* do nothing, just consume the data */
            /*printf("content-l: %"PRIu64", header: %d, recevied: %"PRIu64"\n",
            client->content_length, client->header_size, client->bytes_received);*/

            if (client->chunked) {
                int consume_max;

                str = &client->buffer[client->parser_offset];
                /*printf("parsing chunk: '%s'\n(%"PRIi64" received, %"PRIi64" size, %d parser offset)\n",
                    str, client->chunk_received, client->chunk_size, client->parser_offset
                );*/

                if (client->chunk_size == -1) {
                    /* read chunk size */
                    client->chunk_size = 0;
                    client->chunk_received = 0;
                    end = str + size;

                    for (; str < end; str++) {
                        if (*str == ';' || *str == '\r')
                            break;

                        client->chunk_size *= 16;
                        if (*str >= '0' && *str <= '9')
                            client->chunk_size += *str - '0';
                        else if (*str >= 'A' && *str <= 'Z')
                            client->chunk_size += 10 + *str - 'A';
                        else if (*str >= 'a' && *str <= 'z')
                            client->chunk_size += 10 + *str - 'a';
                        else
                            return 0;
                    }

                    str = strstr(str, "\r\n");
                    if (!str)
                        return 0;
                    str += 2;

                    //printf("---------- chunk size: %"PRIi64", %d read, %d offset, data: '%s'\n", client-
>chunk_size, size, client->parser_offset, str);

                    if (client->chunk_size == 0) {
                        /* chunk of size 0 marks end of content body */
                        client->state = CLIENT_END;
                        client->success = client->status_success ? 1 : 0;
                        return 1;
                    }

                    size -= str - &client->buffer[client->parser_offset];
                    client->parser_offset = str - client->buffer;
                }

                /* consume chunk till chunk_size is reached */
                consume_max = client->chunk_size - client->chunk_received;

                if (size < consume_max)
                    consume_max = size;

                client->chunk_received += consume_max;
                client->parser_offset += consume_max;

                //printf("---------- chunk consuming: %d, received: %"PRIi64" of %"PRIi64", offset: %d\n",
consume_max, client->chunk_received, client->chunk_size, client->parser_offset);

                if (client->chunk_received == client->chunk_size) {
                    if (client->buffer[client->parser_offset] != '\r' || client->buffer[client-
>parser_offset+1] != '\n')
                        return 0;

                    /* got whole chunk, next! */
                    //printf("---------- got whole chunk!!\n");
                    client->chunk_size = -1;
                    client->chunk_received = 0;
                    client->parser_offset += 2;
                    consume_max += 2;

                    /* there is stuff left to parse */
                    if (size - consume_max > 0)
                        return client_parse(client, size - consume_max);
                }

                client->parser_offset = 0;
                client->buffer_offset = 0;

                return 1;
            } else {
                /* not chunked, just consume all data till content-length is reached */
                client->buffer_offset = 0;

                if (client->content_length == -1)
                    return 0;

                if (client->bytes_received == (uint64_t) (client->header_size + client->content_length)) {
                    /* full response received */
                    client->state = CLIENT_END;
                    client->success = client->status_success ? 1 : 0;
                }
            }

            return 1;
    }

    return 1;
}
```

src/client.h - weighttp

Data types defined

- Client

Source code

```
/*
 * weighttp - a lightweight and simple webserver benchmarking tool
 *
 * Author:
 *     Copyright (c) 2009-2011 Thomas Porzelt
 *
 * License:
 *     MIT, see COPYING file
 */

struct Client {
    enum {
        CLIENT_START,
        CLIENT_CONNECTING,
        CLIENT_WRITING,
        CLIENT_READING,
        CLIENT_ERROR,
        CLIENT_END
    } state;

    enum {
        PARSER_START,
        PARSER_HEADER,
        PARSER_BODY
    } parser_state;

    Worker *worker;
    ev_io sock_watcher;
    uint32_t buffer_offset;
    uint32_t parser_offset;
    uint32_t request_offset;
    ev_tstamp ts_start;
    ev_tstamp ts_end;
    uint8_t keepalive;
    uint8_t success;
    uint8_t status_success;
    uint8_t chunked;
    int64_t chunk_size;
    int64_t chunk_received;
    int64_t content_length;
    uint64_t bytes_received; /* including http header */
    uint16_t header_size;

    char buffer[CLIENT_BUFFER_SIZE];
};

Client *client_new(Worker *worker);
void client_free(Client *client);
void client_state_machine(Client *client);
```

src/worker.h - weighttp

Data types defined

- Stats
- Worker

Source code

```c
/*
* weighttp - a lightweight and simple webserver benchmarking tool
*
* Author:
*     Copyright (c) 2009-2011 Thomas Porzelt
*
* License:
*     MIT, see COPYING file
*/

struct Stats {
    ev_tstamp req_ts_min;    /* minimum time taken for a request */
    ev_tstamp req_ts_max;    /* maximum time taken for a request */
    ev_tstamp req_ts_total;    /* total time taken for all requests (this is not ts_end - ts_start!) */
    uint64_t req_todo;        /* total number of requests to do */
    uint64_t req_started;    /* total number of requests started */
    uint64_t req_done;        /* total number of requests done */
    uint64_t req_success;    /* total number of successful requests */
    uint64_t req_failed;    /* total number of failed requests */
    uint64_t req_error;        /* total number of error'd requests */
    uint64_t bytes_total;    /* total number of bytes received (html+body) */
    uint64_t bytes_body;    /* total number of bytes received (body) */
    uint64_t req_1xx;
    uint64_t req_2xx;
    uint64_t req_3xx;
    uint64_t req_4xx;
    uint64_t req_5xx;
};

struct Worker {
    uint8_t id;
    Config *config;
    struct ev_loop *loop;
    char *request;
    Client **clients;
    uint16_t num_clients;
    Stats stats;
    uint64_t progress_interval;
};


Worker *worker_new(uint8_t id, Config *config, uint16_t num_clients, uint64_t num_requests);
void worker_free(Worker *worker);
void *worker_thread(void* arg);
```

src/weighttp.h - weighttp

Data types defined

- Client
- Config
- Config
- Stats
- Worker

Macros defined

- CLIENT_BUFFER_SIZE
- UNUSED
- WEIGHTTP_H
- W_ERROR
- W_MALLOC
- W_REALLOC

Source code

```c
/*
* weighttp - a lightweight and simple webserver benchmarking tool
*
* Author:
*     Copyright (c) 2009-2011 Thomas Porzelt
*
* License:
*     MIT, see COPYING file
*/

 #ifndef WEIGHTTP_H
 #define WEIGHTTP_H 1

 #include <stdio.h>
 #include <stdlib.h>
 #include <time.h>
 #include <errno.h>
 #include <string.h>

 #include <unistd.h>
 #include <stdint.h>
 #include <fcntl.h>
 #include <inttypes.h>
 #include <sys/socket.h>
 #include <netdb.h>

 #include <ev.h>
 #include <pthread.h>

 #define CLIENT_BUFFER_SIZE 32 * 1024

 #define W_MALLOC(t, n) ((t*) calloc((n), sizeof(t)))
 #define W_REALLOC(p, t, n) ((t*) realloc(p, (n) * sizeof(t)))
 #define W_ERROR(f, ...) fprintf(stderr, "error: " f "\n", __VA_ARGS__)
 #define UNUSED(x) ( (void)(x) )

```

```
struct Config;
typedef struct Config Config;
struct Stats;
typedef struct Stats Stats;
struct Worker;
typedef struct Worker Worker;
struct Client;
typedef struct Client Client;

#include "client.h"
#include "worker.h"


struct Config {
    uint64_t req_count;
    uint8_t thread_count;
    uint16_t concur_count;
    uint8_t keep_alive;

    char *request;
    uint32_t request_size;
    struct addrinfo *saddr;
};

uint64_t str_to_uint64(char *str);

#endif
```

src/weighttp.c - weighttp

Functions defined

Source code

```
/*
* weighttp - a lightweight and simple webserver benchmarking tool
*
* Author:
*     Copyright (c) 2009-2011 Thomas Porzelt
*
* License:
*     MIT, see COPYING file
*/

 #include "weighttp.h"

 extern int optind, optopt; /* getopt */

 static void show_help(void) {
     printf("weighttp <options> <url>\n");
     printf("  -n num   number of requests    (mandatory)\n");
     printf("  -t num   threadcount           (default: 1)\n");
     printf("  -c num   concurrent clients    (default: 1)\n");
     printf("  -k       keep alive            (default: no)\n");
     printf("  -6       use ipv6              (default: no)\n");
     printf("  -H str   add header to request\n");
     printf("  -h       show help and exit\n");
     printf("  -v       show version and exit\n\n");
     printf("example: weighttpd -n 10000 -c 10 -t 2 -k -H \"User-Agent: foo\" localhost/index.html\n\n");
 }

 static struct addrinfo *resolve_host(char *hostname, uint16_t port, uint8_t use_ipv6) {
     int err;
     char port_str[6];
     struct addrinfo hints, *res, *res_first, *res_last;

     memset(&hints, 0, sizeof(hints));
     hints.ai_family = PF_UNSPEC;
     hints.ai_socktype = SOCK_STREAM;

     sprintf(port_str, "%d", port);

     err = getaddrinfo(hostname, port_str, &hints, &res_first);

     if (err) {
         W_ERROR("could not resolve hostname: %s", hostname);
         return NULL;
     }

     /* search for an ipv4 address, no ipv6 yet */
     res_last = NULL;
     for (res = res_first; res != NULL; res = res->ai_next) {
         if (res->ai_family == AF_INET && !use_ipv6)
             break;
         else if (res->ai_family == AF_INET6 && use_ipv6)
             break;

```

```c
        res_last = res;
    }

    if (!res) {
        freeaddrinfo(res_first);
        W_ERROR("could not resolve hostname: %s", hostname);
        return NULL;
    }

    if (res != res_first) {
        /* unlink from list and free rest */
        res_last->ai_next = res->ai_next;
        freeaddrinfo(res_first);
        res->ai_next = NULL;
    }

    return res;
}

static char *forge_request(char *url, char keep_alive, char **host, uint16_t *port, char **headers, uint8_t headers_num) {
    char *c, *end;
    char *req;
    uint32_t len;
    uint8_t i;
    uint8_t have_user_agent, have_host;

    *host = NULL;
    *port = 0;

    if (strncmp(url, "http://", 7) == 0)
        url += 7;
    else if (strncmp(url, "https://", 8) == 0) {
        W_ERROR("%s", "no ssl support yet");
        url += 8;
        return NULL;
    }

    len = strlen(url);

    if ((c = strchr(url, ':'))) {
        /* found ':' => host:port */
        *host = W_MALLOC(char, c - url + 1);
        memcpy(*host, url, c - url);
        (*host)[c - url] = '\0';

        if ((end = strchr(c+1, '/'))) {
            *end = '\0';
            *port = atoi(c+1);
            *end = '/';
            url = end;
         } else {
            *port = atoi(c+1);
            url += len;
         }
     } else {
         *port = 80;

         if ((c = strchr(url, '/'))) {
             *host = W_MALLOC(char, c - url + 1);
             memcpy(*host, url, c - url);
             (*host)[c - url] = '\0';
             url = c;
         } else {
             *host = W_MALLOC(char, len + 1);
             memcpy(*host, url, len);
             (*host)[len] = '\0';
             url += len;
         }
     }

     if (*port == 0) {
         W_ERROR("%s", "could not parse url");
         free(*host);
         return NULL;
     }

    if (*url == '\0')
        url = "/";

    // total request size
    len = strlen("GET HTTP/1.1\r\nConnection: keep-alive\r\n\r\n") + 1;
    len += strlen(url);

    have_user_agent = 0;
    have_host = 0;
    for (i = 0; i < headers_num; i++) {
        len += strlen(headers[i]) + strlen("\r\n");
        if (strncmp(headers[i], "User-Agent: ", sizeof("User-Agent: ")-1) == 0)
            have_user_agent = 1;
        if (strncasecmp(headers[i], "Host:", sizeof("Host:")-1) == 0)
            have_host = 1;
    }

    if (!have_user_agent)
        len += strlen("User-Agent: weighttp/" VERSION "\r\n");

    if (!have_host) {
        len += strlen("Host: :65536\r\n")-1;
        len += strlen(*host);
    }

    req = W_MALLOC(char, len);

    strcpy(req, "GET ");
    strcat(req, url);
    strcat(req, " HTTP/1.1\r\n");

    if (!have_host) {
        strcat(req, "Host: ");
        strcat(req, *host);
        if (*port != 80)
            sprintf(req + strlen(req), ":%"PRIu16, *port);
        strcat(req, "\r\n");
    }

    if (!have_user_agent)
        sprintf(req + strlen(req), "User-Agent: weighttp/" VERSION "\r\n");

    for (i = 0; i < headers_num; i++) {
        strcat(req, headers[i]);
        strcat(req, "\r\n");
    }

    if (keep_alive)
        strcat(req, "Connection: keep-alive\r\n\r\n");
    else
        strcat(req, "Connection: close\r\n\r\n");

    return req;
}

uint64_t str_to_uint64(char *str) {
    uint64_t i;

    for (i = 0; *str; str++) {
        if (*str < '0' || *str > '9')
            return UINT64_MAX;

        i *= 10;
        i += *str - '0';
    }

    return i;
}

int main(int argc, char *argv[]) {
    Worker **workers;
    pthread_t *threads;
    int i;
    char c;
    int err;
```

```c
	struct ev_loop *loop;
	ev_tstamp ts_start, ts_end;
	Config config;
	Worker *worker;
	char *host;
	uint16_t port;
	uint8_t use_ipv6;
	uint16_t rest_concur, rest_req;
	Stats stats;
	ev_tstamp duration;
	int sec, millisec, microsec;
	uint64_t rps;
	uint64_t kbps;
	char **headers;
	uint8_t headers_num;

	printf("weighttp - a lightweight and simple webserver benchmarking tool\n\n");

	headers = NULL;
	headers_num = 0;

	/* default settings */
	use_ipv6 = 0;
	config.thread_count = 1;
	config.concur_count = 1;
	config.req_count = 0;
	config.keep_alive = 0;

	while ((c = getopt(argc, argv, ":hv6kn:t:c:H:")) != -1) {
		switch (c) {
			case 'h':
				show_help();
				return 0;
			case 'v':
				printf("version:    " VERSION "\n");
				printf("build-date: " __DATE__ " " __TIME__ "\n\n");
				return 0;
			case '6':
				use_ipv6 = 1;
				break;
			case 'k':
				config.keep_alive = 1;
				break;
			case 'n':
				config.req_count = str_to_uint64(optarg);
				break;
			case 't':
				config.thread_count = atoi(optarg);
				break;
			case 'c':
				config.concur_count = atoi(optarg);
				break;
			case 'H':
				headers = W_REALLOC(headers, char*, headers_num+1);
				headers[headers_num] = optarg;
				headers_num++;
				break;
			case '?':
				W_ERROR("unkown option: -%c", optopt);
				show_help();
				return 1;
		}
	}

	if ((argc - optind) < 1) {
		W_ERROR("%s", "missing url argument\n");
		show_help();
		return 1;
	} else if ((argc - optind) > 1) {
		W_ERROR("%s", "too many arguments\n");
		show_help();
		return 1;
	}

	/* check for sane arguments */
	if (!config.thread_count) {
        W_ERROR("%s", "thread count has to be > 0\n");
        show_help();
        return 1;
    }
    if (!config.concur_count) {
        W_ERROR("%s", "number of concurrent clients has to be > 0\n");
        show_help();
        return 1;
    }
    if (!config.req_count) {
        W_ERROR("%s", "number of requests has to be > 0\n");
        show_help();
        return 1;
    }
    if (config.req_count == UINT64_MAX || config.thread_count > config.req_count || config.thread_count > 
config.concur_count || config.concur_count > config.req_count) {
        W_ERROR("%s", "insane arguments\n");
        show_help();
        return 1;
    }


    loop = ev_default_loop(0);
    if (!loop) {
        W_ERROR("%s", "could not initialize libev\n");
        return 2;
    }

    if (NULL == (config.request = forge_request(argv[optind], config.keep_alive, &host, &port, headers, 
headers_num))) {
        return 1;
    }

    config.request_size = strlen(config.request);
    //printf("Request (%d):\n==========\n%s==========\n", config.request_size, config.request);
    //printf("host: '%s', port: %d\n", host, port);

    /* resolve hostname */
    if(!(config.saddr = resolve_host(host, port, use_ipv6))) {
        return 1;
    }

    /* spawn threads */
    threads = W_MALLOC(pthread_t, config.thread_count);
    workers = W_MALLOC(Worker*, config.thread_count);

    rest_concur = config.concur_count % config.thread_count;
    rest_req = config.req_count % config.thread_count;

    printf("starting benchmark...\n");

    memset(&stats, 0, sizeof(stats));
    ts_start = ev_time();

    for (i = 0; i < config.thread_count; i++) {
        uint64_t reqs = config.req_count / config.thread_count;
        uint16_t concur = config.concur_count / config.thread_count;
        uint64_t diff;

        if (rest_concur) {
            diff = (i == config.thread_count) ? rest_concur : (rest_concur / config.thread_count);
            diff = diff ? diff : 1;
            concur += diff;
            rest_concur -= diff;
        }

        if (rest_req) {
            diff = (i == config.thread_count) ? rest_req : (rest_req / config.thread_count);
            diff = diff ? diff : 1;
            reqs += diff;
            rest_req -= diff;
        }
        printf("spawning thread #%d: %"PRIu16" concurrent requests, %"PRIu64" total requests\n", i+1, concur, 
reqs);
        workers[i] = worker_new(i+1, &config, concur, reqs);

        if (!(workers[i])) {
            W_ERROR("%s", "failed to allocate worker or client");
            return 1;
        }

        err = pthread_create(&threads[i], NULL, worker_thread, (void*)workers[i]);

        if (err != 0) {
            W_ERROR("failed spawning thread (%d)", err);
            return 2;
        }
    }

    for (i = 0; i < config.thread_count; i++) {
        err = pthread_join(threads[i], NULL);
        worker = workers[i];

        if (err != 0) {
            W_ERROR("failed joining thread (%d)", err);
            return 3;
        }

        stats.req_started += worker->stats.req_started;
        stats.req_done += worker->stats.req_done;
        stats.req_success += worker->stats.req_success;
        stats.req_failed += worker->stats.req_failed;
        stats.bytes_total += worker->stats.bytes_total;
        stats.bytes_body += worker->stats.bytes_body;
        stats.req_2xx += worker->stats.req_2xx;
        stats.req_3xx += worker->stats.req_3xx;
        stats.req_4xx += worker->stats.req_4xx;
        stats.req_5xx += worker->stats.req_5xx;

        worker_free(worker);
    }

    ts_end = ev_time();
    duration = ts_end - ts_start;
    sec = duration;
    duration -= sec;
    duration = duration * 1000;
    millisec = duration;
    duration -= millisec;
    microsec = duration * 1000;
    rps = stats.req_done / (ts_end - ts_start);
    kbps = stats.bytes_total / (ts_end - ts_start) / 1024;
    printf("\nfinished in %d sec, %d millisec and %d microsec, %"PRIu64" req/s, %"PRIu64" kbyte/s\n", sec, millisec, microsec, rps, kbps);
    printf("requests: %"PRIu64" total, %"PRIu64" started, %"PRIu64" done, %"PRIu64" succeeded, %"PRIu64" failed, %"PRIu64" errored\n",
        config.req_count, stats.req_started, stats.req_done, stats.req_success, stats.req_failed, stats.req_error
    );
    printf("status codes: %"PRIu64" 2xx, %"PRIu64" 3xx, %"PRIu64" 4xx, %"PRIu64" 5xx\n",
        stats.req_2xx, stats.req_3xx, stats.req_4xx, stats.req_5xx
    );
    printf("traffic: %"PRIu64" bytes total, %"PRIu64" bytes http, %"PRIu64" bytes data\n",
        stats.bytes_total,  stats.bytes_total - stats.bytes_body, stats.bytes_body
    );

    ev_default_destroy();

    free(threads);
    free(workers);
    free(config.request);
    free(host);
    free(headers);
    freeaddrinfo(config.saddr);

    return 0;
}
```

One Level Up   Top Level

src/worker.c - weighttp

Functions defined

- worker_free
- worker_new
- worker_thread

Source code

```c
/*
* weighttp - a lightweight and simple webserver benchmarking tool
*
* Author:
*     Copyright (c) 2009-2011 Thomas Porzelt
*
* License:
*     MIT, see COPYING file
*/

#include "weighttp.h"

Worker *worker_new(uint8_t id, Config *config, uint16_t num_clients, uint64_t num_requests) {
    Worker *worker;
    uint16_t i;

    worker = W_MALLOC(Worker, 1);
    worker->id = id;
    worker->loop = ev_loop_new(0);
    ev_ref(worker->loop);
    worker->config = config;
    worker->num_clients = num_clients;
    worker->stats.req_todo = num_requests;
    worker->progress_interval = num_requests / 10;

    if (worker->progress_interval == 0)
        worker->progress_interval = 1;

    worker->clients = W_MALLOC(Client*, num_clients);

    for (i = 0; i < num_clients; i++) {
        if (NULL == (worker->clients[i] = client_new(worker)))
            return NULL;
    }

    return worker;
}

void worker_free(Worker *worker) {
    uint16_t i;

    for (i = 0; i < worker->num_clients; i++)
        client_free(worker->clients[i]);

    free(worker->clients);
    free(worker);
}

void *worker_thread(void* arg) {
    uint16_t i;
    Worker *worker = (Worker*)arg;

    /* start all clients */
    for (i = 0; i < worker->num_clients; i++) {
        if (worker->stats.req_started < worker->stats.req_todo)
            client_state_machine(worker->clients[i]);
    }

    ev_loop(worker->loop, 0);

    ev_loop_destroy(worker->loop);

    return NULL;
}
```

One Level Up Top Level