

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

```
1 /*
2  * weighttp - a lightweight and simple webserver benchmarking tool
3  *
4  * Author:
5  *   Copyright (c) 2009-2011 Thomas Porzelt
6  *
7  * License:
8  *   MIT, see COPYING file
9  */
10
11 #include "weighttp.h"
12
13 static uint8_t client_parse(Client *client, int size);
14 static void client_io_cb(struct ev_loop *loop, ev_io *w, int revents);
15 static void client_set_events(Client *client, int events);
16 /*
17  static void client_add_events(Client *client, int events);
18  static void client_rem_events(Client *client, int events);
19
20  static void client_add_events(Client *client, int events) {
21      struct ev_loop *loop = client->worker->loop;
22      ev_io *watcher = &client->sock_watcher;
23
24      if ((watcher->events & events) == events)
25          return;
26
27      ev_io_stop(loop, watcher);
28      ev_io_set(watcher, watcher->fd, watcher->events | events);
29      ev_io_start(loop, watcher);
30  }
31
32  static void client_rem_events(Client *client, int events) {
33      struct ev_loop *loop = client->worker->loop;
34      ev_io *watcher = &client->sock_watcher;
35
36      if (0 == (watcher->events & events))
37          return;
38
39      ev_io_stop(loop, watcher);
40      ev_io_set(watcher, watcher->fd, watcher->events & ~events);
41      ev_io_start(loop, watcher);
42  }
43  */
44
45 static void client_set_events(Client *client, int events) {
46     struct ev_loop *loop = client->worker->loop;
47     ev_io *watcher = &client->sock_watcher;
```

```

48     if (events == (watcher->events & (EV_READ | EV_WRITE)))
49         return;
50
51
52     ev_io_stop(loop, watcher);
53     ev_io_set(watcher, watcher->fd, (watcher->events & ~(EV_READ | EV_WRITE)) | events);
54     ev_io_start(loop, watcher);
55 }
56
57 Client *client_new(Worker *worker) {
58     Client *client;
59
60     client = W_MALLOC(Client, 1);
61     client->state = CLIENT_START;
62     client->worker = worker;
63     client->sock_watcher.fd = -1;
64     client->sock_watcher.data = client;
65     client->content_length = -1;
66     client->buffer_offset = 0;
67     client->request_offset = 0;
68     client->keepalive = client->worker->config->keep_alive;
69     client->chunked = 0;
70     client->chunk_size = -1;
71     client->chunk_received = 0;
72
73     return client;
74 }
75
76 void client_free(Client *client) {
77     if (client->sock_watcher.fd != -1) {
78         ev_io_stop(client->worker->loop, &client->sock_watcher);
79         shutdown(client->sock_watcher.fd, SHUT_WR);
80         close(client->sock_watcher.fd);
81     }
82
83     free(client);
84 }
85
86 static void client_reset(Client *client) {
87     //printf("keep alive: %d\n", client->keepalive);
88     if (!client->keepalive) {
89         if (client->sock_watcher.fd != -1) {
90             ev_io_stop(client->worker->loop, &client->sock_watcher);
91             shutdown(client->sock_watcher.fd, SHUT_WR);
92             close(client->sock_watcher.fd);
93             client->sock_watcher.fd = -1;
94         }
95
96         client->state = CLIENT_START;
97     } else {
98         client_set_events(client, EV_WRITE);
99         client->state = CLIENT_WRITING;
100        client->worker->stats.req_started++;
101    }
102
103    client->parser_state = PARSER_START;
104    client->buffer_offset = 0;
105    client->parser_offset = 0;
106    client->request_offset = 0;
107    client->ts_start = 0;
108    client->ts_end = 0;
109    client->status_success = 0;
110    client->success = 0;
111    client->content_length = -1;
112    client->bytes_received = 0;
113    client->header_size = 0;
114    client->keepalive = client->worker->config->keep_alive;
115    client->chunked = 0;
116    client->chunk_size = -1;
117    client->chunk_received = 0;
118 }
119
120 static uint8_t client_connect(Client *client) {
121     //printf("connecting...\n");
122     start:
123

```

```

124     if (-1 == connect(client->sock_watcher.fd, client->worker->config->saddr->ai_addr, client->worker-
>config->saddr->ai_addrlen)) {
125         switch (errno) {
126             case EINPROGRESS:
127             case EALREADY:
128                 /* async connect now in progress */
129                 client->state = CLIENT_CONNECTING;
130                 return 1;
131             case EISCONN:
132                 break;
133             case EINTR:
134                 goto start;
135             default:
136                 {
137                     strerror_r(errno, client->buffer, sizeof(client->buffer));
138                     W_ERROR("connect() failed: %s (%d)", client->buffer, errno);
139                     return 0;
140                 }
141         }
142     }
143
144     /* successfully connected */
145     client->state = CLIENT_WRITING;
146     return 1;
147 }
148
149 static void client_io_cb(struct ev_loop *loop, ev_io *w, int revents) {
150     Client *client = w->data;
151
152     UNUSED(loop);
153     UNUSED(revents);
154
155     client_state_machine(client);
156 }
157
158 void client_state_machine(Client *client) {
159     int r;
160     Config *config = client->worker->config;
161
162     start:
163     //printf("state: %d\n", client->state);
164     switch (client->state) {
165         case CLIENT_START:
166             client->worker->stats.req_started++;
167
168             do {
169                 r = socket(config->saddr->ai_family, config->saddr->ai_socktype, config->saddr->ai_protocol);
170             } while (-1 == r && errno == EINTR);
171
172             if (-1 == r) {
173                 client->state = CLIENT_ERROR;
174                 strerror_r(errno, client->buffer, sizeof(client->buffer));
175                 W_ERROR("socket() failed: %s (%d)", client->buffer, errno);
176                 goto start;
177             }
178
179             /* set non-blocking */
180             fcntl(r, F_SETFL, O_NONBLOCK | O_RDWR);
181
182             ev_init(&client->sock_watcher, client_io_cb);
183             ev_io_set(&client->sock_watcher, r, EV_WRITE);
184             ev_io_start(client->worker->loop, &client->sock_watcher);
185
186             if (!client_connect(client)) {
187                 client->state = CLIENT_ERROR;
188                 goto start;
189             } else {
190                 client_set_events(client, EV_WRITE);
191                 return;
192             }
193         case CLIENT_CONNECTING:
194             if (!client_connect(client)) {
195                 client->state = CLIENT_ERROR;
196                 goto start;
197             }
198         case CLIENT_WRITING:

```

```

199     while (1) {
200         r = write(client->sock_watcher.fd, &config->request[client->request_offset], config-
>request_size - client->request_offset);
201         //printf("write(%d - %d = %d): %d\n", config->request_size, client->request_offset, config-
>request_size - client->request_offset, r);
202         if (r == -1) {
203             /* error */
204             if (errno == EINTR)
205                 continue;
206             strerror_r(errno, client->buffer, sizeof(client->buffer));
207             W_ERROR("write() failed: %s (%d)", client->buffer, errno);
208             client->state = CLIENT_ERROR;
209             goto start;
210         } else if (r != 0) {
211             /* success */
212             client->request_offset += r;
213             if (client->request_offset == config->request_size) {
214                 /* whole request was sent, start reading */
215                 client->state = CLIENT_READING;
216                 client_set_events(client, EV_READ);
217             }
218
219             return;
220         } else {
221             /* disconnect */
222             client->state = CLIENT_END;
223             goto start;
224         }
225     }
226 case CLIENT_READING:
227     while (1) {
228         r = read(client->sock_watcher.fd, &client->buffer[client->buffer_offset], sizeof(client-
>buffer) - client->buffer_offset - 1);
229         //printf("read(): %d, offset was: %d\n", r, client->buffer_offset);
230         if (r == -1) {
231             /* error */
232             if (errno == EINTR)
233                 continue;
234             strerror_r(errno, client->buffer, sizeof(client->buffer));
235             W_ERROR("read() failed: %s (%d)", client->buffer, errno);
236             client->state = CLIENT_ERROR;
237         } else if (r != 0) {
238             /* success */
239             client->bytes_received += r;
240             client->buffer_offset += r;
241             client->worker->stats.bytes_total += r;
242
243             if (client->buffer_offset >= sizeof(client->buffer)) {
244                 /* too big response header */
245                 client->state = CLIENT_ERROR;
246                 break;
247             }
248             client->buffer[client->buffer_offset] = '\0';
249             //printf("buffer:\n=====\n%s\n=====\n", client->buffer);
250             if (!client_parse(client, r)) {
251                 client->state = CLIENT_ERROR;
252                 //printf("parser failed\n");
253                 break;
254             } else {
255                 if (client->state == CLIENT_END)
256                     goto start;
257                 else
258                     return;
259             }
260         } else {
261             /* disconnect */
262             if (client->parser_state == PARSER_BODY && !client->keepalive && client->status_success
&& !client->chunked && client->content_length == -1) {
263                 client->success = 1;
264                 client->state = CLIENT_END;
265             } else {
266                 client->state = CLIENT_ERROR;
267             }
268         }
269         goto start;
270     }
271 }

```

```

272     }
273
274     case CLIENT_ERROR:
275         //printf("client error\n");
276         client->worker->stats.req_error++;
277         client->keepalive = 0;
278         client->success = 0;
279         client->state = CLIENT_END;
280     case CLIENT_END:
281         /* update worker stats */
282         client->worker->stats.req_done++;
283
284         if (client->success) {
285             client->worker->stats.req_success++;
286             client->worker->stats.bytes_body += client->bytes_received - client->header_size;
287         } else {
288             client->worker->stats.req_failed++;
289         }
290
291         /* print progress every 10% done */
292         if (client->worker->id == 1 && client->worker->stats.req_done % client->worker->progress_interval
== 0) {
293             printf("progress: %3d%% done\n",
294                 (int) (client->worker->stats.req_done * 100 / client->worker->stats.req_todo)
295             );
296         }
297
298         if (client->worker->stats.req_started == client->worker->stats.req_todo) {
299             /* this worker has started all requests */
300             client->keepalive = 0;
301             client\_reset(client);
302
303             if (client->worker->stats.req_done == client->worker->stats.req_todo) {
304                 /* this worker has finished all requests */
305                 ev_unref(client->worker->loop);
306             }
307         } else {
308             client\_reset(client);
309             goto start;
310         }
311     }
312 }
313
314
315 static uint8_t client\_parse(Client *client, int size) {
316     char *end, *str;
317     uint16_t status_code;
318
319     switch (client->parser_state) {
320     case PARSER\_START:
321         //printf("parse (START):\n%s\n", &client->buffer[client->parser_offset]);
322         /* look for HTTP/1.1 200 OK */
323         if (client->buffer_offset < sizeof("HTTP/1.1 200\r\n"))
324             return 1;
325
326         if (strncmp(client->buffer, "HTTP/1.1 ", sizeof("HTTP/1.1 ") - 1) != 0)
327             return 0;
328
329         /* now the status code
330         status_code = 0;
331         str = client->buffer + sizeof("HTTP/1.1 ") - 1;
332         for (end = str + 3; str != end; str++) {
333             if (*str < '0' || *str > '9')
334                 return 0;
335
336             status_code *= 10;
337             status_code += *str - '0';
338         }
339
340         if (status_code >= 200 && status_code < 300) {
341             client->worker->stats.req_2xx++;
342             client->status_success = 1;
343         } else if (status_code < 400) {
344             client->worker->stats.req_3xx++;
345             client->status_success = 1;
346         } else if (status_code < 500) {

```

```

347     client->worker->stats.req_4xx++;
348 } else if (status_code < 600) {
349     client->worker->stats.req_5xx++;
350 } else {
351     // invalid status code
352     return 0;
353 }
354
355 // look for next \r\n
356 end = strchr(end, '\r');
357 if (!end || *(end+1) != '\n')
358     return 0;
359
360 client->parser_offset = end + 2 - client->buffer;
361 client->parser_state = PARSER_HEADER;
362 case PARSER_HEADER:
363     //printf("parse (HEADER)\n");
364     /* look for Content-Length and Connection header */
365     while (NULL != (end = strchr(&client->buffer[client->parser_offset], '\r'))) {
366         if (*(end+1) != '\n')
367             return 0;
368
369         if (end == &client->buffer[client->parser_offset]) {
370             /* body reached */
371             client->parser_state = PARSER_BODY;
372             client->header_size = end + 2 - client->buffer;
373             //printf("body reached\n");
374
375             return client_parse(client, size - client->header_size);
376         }
377
378         *end = '\0';
379         str = &client->buffer[client->parser_offset];
380         //printf("checking header: '%s'\n", str);
381
382         if (strncmp(str, "Content-Length: ", sizeof("Content-Length: ") - 1) == 0) {
383             /* content length header */
384             client->content_length = str_to_uint64(str + sizeof("Content-Length: ") - 1);
385         } else if (strncmp(str, "Connection: ", sizeof("Connection: ") - 1) == 0) {
386             /* connection header */
387             str += sizeof("Connection: ") - 1;
388
389             if (strncmp(str, "close", sizeof("close") - 1) == 0)
390                 client->keepalive = 0;
391             else if (strncmp(str, "Keep-Alive", sizeof("Keep-Alive") - 1) == 0)
392                 client->keepalive = client->worker->config->keep_alive;
393             else if (strncmp(str, "keep-alive", sizeof("keep-alive") - 1) == 0)
394                 client->keepalive = client->worker->config->keep_alive;
395             else
396                 return 0;
397         } else if (strncmp(str, "Transfer-Encoding: ", sizeof("Transfer-Encoding: ") - 1) == 0) {
398             /* transfer encoding header */
399             str += sizeof("Transfer-Encoding: ") - 1;
400
401             if (strncmp(str, "chunked", sizeof("chunked") - 1) == 0)
402                 client->chunked = 1;
403             else
404                 return 0;
405         }
406
407         if (*(end+2) == '\r' && *(end+3) == '\n') {
408             /* body reached */
409             client->parser_state = PARSER_BODY;
410             client->header_size = end + 4 - client->buffer;
411             client->parser_offset = client->header_size;
412             //printf("body reached\n");
413
414             return client_parse(client, size - client->header_size);
415         }
416
417         client->parser_offset = end - client->buffer + 2;
418     }
419 }
420
421 return 1;
422 case PARSER_BODY:

```

```

423 //printf("parse (BODY)\n");
424 /* do nothing, just consume the data */
425 /*printf("content-1: %"PRIu64", header: %d, received: %"PRIu64"\n",
426 client->content_length, client->header_size, client->bytes_received);*/
427
428 if (client->chunked) {
429     int consume_max;
430
431     str = &client->buffer[client->parser_offset];
432     /*printf("parsing chunk: '%s'\n(%"PRIi64" received, %"PRIi64" size, %d parser offset)\n",
433         str, client->chunk_received, client->chunk_size, client->parser_offset
434     );*/
435
436     if (client->chunk_size == -1) {
437         /* read chunk size */
438         client->chunk_size = 0;
439         client->chunk_received = 0;
440         end = str + size;
441
442         for (; str < end; str++) {
443             if (*str == ';' || *str == '\r')
444                 break;
445
446             client->chunk_size *= 16;
447             if (*str >= '0' && *str <= '9')
448                 client->chunk_size += *str - '0';
449             else if (*str >= 'A' && *str <= 'Z')
450                 client->chunk_size += 10 + *str - 'A';
451             else if (*str >= 'a' && *str <= 'z')
452                 client->chunk_size += 10 + *str - 'a';
453             else
454                 return 0;
455         }
456
457         str = strstr(str, "\r\n");
458         if (!str)
459             return 0;
460         str += 2;
461
462         /*printf("----- chunk size: %"PRIi64", %d read, %d offset, data: '%s'\n", client-
463 >chunk_size, size, client->parser_offset, str);
464
465         if (client->chunk_size == 0) {
466             /* chunk of size 0 marks end of content body */
467             client->state = CLIENT_END;
468             client->success = client->status_success ? 1 : 0;
469             return 1;
470         }
471
472         size -= str - &client->buffer[client->parser_offset];
473         client->parser_offset = str - client->buffer;
474     }
475
476     /* consume chunk till chunk_size is reached */
477     consume_max = client->chunk_size - client->chunk_received;
478
479     if (size < consume_max)
480         consume_max = size;
481
482     client->chunk_received += consume_max;
483     client->parser_offset += consume_max;
484
485     /*printf("----- chunk consuming: %d, received: %"PRIi64" of %"PRIi64", offset: %d\n",
486 consume_max, client->chunk_received, client->chunk_size, client->parser_offset);
487
488     if (client->chunk_received == client->chunk_size) {
489         if (client->buffer[client->parser_offset] != '\r' || client->buffer[client-
490 >parser_offset+1] != '\n')
491             return 0;
492
493         /* got whole chunk, next! */
494         /*printf("----- got whole chunk!!\n");
495         client->chunk_size = -1;
496         client->chunk_received = 0;
497         client->parser_offset += 2;
498         consume_max += 2;

```



```
496         /* there is stuff left to parse */
497         if (size - consume_max > 0)
498             return client\_parse(client, size - consume_max);
499     }
500
501     client->parser_offset = 0;
502     client->buffer_offset = 0;
503
504     return 1;
505 } else {
506     /* not chunked, just consume all data till content-length is reached */
507     client->buffer_offset = 0;
508
509     if (client->content_length == -1)
510         return 0;
511
512     if (client->bytes_received == (uint64_t) (client->header_size + client->content_length)) {
513         /* full response received */
514         client->state = CLIENT_END;
515         client->success = client->status_success ? 1 : 0;
516     }
517 }
518
519     return 1;
520 }
521
522 return 1;
523 }
524 }
```

[One Level Up](#)

[Top Level](#)

src/client.h - weighttp

Data types defined

- [Client](#)

Source code

```
1  /*
2  * weighttp - a lightweight and simple webserver benchmarking tool
3  *
4  * Author:
5  *     Copyright (c) 2009-2011 Thomas Porzelt
6  *
7  * License:
8  *     MIT, see COPYING file
9  */
10
11 struct Client {
12     enum {
13         CLIENT_START,
14         CLIENT_CONNECTING,
15         CLIENT_WRITING,
16         CLIENT_READING,
17         CLIENT_ERROR,
18         CLIENT_END
19     } state;
20
21     enum {
22         PARSER_START,
23         PARSER_HEADER,
24         PARSER_BODY
25     } parser_state;
26
27     Worker *worker;
28     ev_io sock_watcher;
29     uint32_t buffer_offset;
30     uint32_t parser_offset;
31     uint32_t request_offset;
32     ev_tstamp ts_start;
33     ev_tstamp ts_end;
34     uint8_t keepalive;
35     uint8_t success;
36     uint8_t status_success;
37     uint8_t chunked;
38     int64_t chunk_size;
39     int64_t chunk_received;
40     int64_t content_length;
41     uint64_t bytes_received; /* including http header */
42     uint16_t header_size;
43
44     char buffer[CLIENT\_BUFFER\_SIZE];
45 };
46
47 Client *client_new(Worker *worker);
48 void client_free(Client *client);
49 void client_state_machine(Client *client);
```

src/worker.h - weighttp

Data types defined

- [Stats](#)
- [Worker](#)

Source code

```
1  /*
2  * weighttp - a lightweight and simple webserver benchmarking tool
3  *
4  * Author:
5  *   Copyright (c) 2009-2011 Thomas Porzelt
6  *
7  * License:
8  *   MIT, see COPYING file
9  */
10
11 struct Stats {
12     ev_tstamp req_ts_min;    /* minimum time taken for a request */
13     ev_tstamp req_ts_max;    /* maximum time taken for a request */
14     ev_tstamp req_ts_total;  /* total time taken for all requests (this is not ts_end - ts_start!) */
15     uint64_t req_todo;       /* total number of requests to do */
16     uint64_t req_started;    /* total number of requests started */
17     uint64_t req_done;       /* total number of requests done */
18     uint64_t req_success;    /* total number of successful requests */
19     uint64_t req_failed;     /* total number of failed requests */
20     uint64_t req_error;      /* total number of error'd requests */
21     uint64_t bytes_total;    /* total number of bytes received (html+body) */
22     uint64_t bytes_body;     /* total number of bytes received (body) */
23     uint64_t req_1xx;
24     uint64_t req_2xx;
25     uint64_t req_3xx;
26     uint64_t req_4xx;
27     uint64_t req_5xx;
28 };
29
30 struct Worker {
31     uint8_t id;
32     Config *config;
33     struct ev_loop *loop;
34     char *request;
35     Client **clients;
36     uint16_t num_clients;
37     Stats stats;
38     uint64_t progress_interval;
39 };
40
41
42 Worker *worker_new(uint8_t id, Config *config, uint16_t num_clients, uint64_t num_requests);
43 void worker_free(Worker *worker);
44 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

```
1  /*
2  * weighttp - a lightweight and simple webserver benchmarking tool
3  *
4  * Author:
5  *     Copyright (c) 2009-2011 Thomas Porzelt
6  *
7  * License:
8  *     MIT, see COPYING file
9  */
10
11 #ifndef WEIGHTTP_H
12 #define WEIGHTTP_H 1
13
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <time.h>
17 #include <errno.h>
18 #include <string.h>
19
20 #include <unistd.h>
21 #include <stdint.h>
22 #include <fcntl.h>
23 #include <inttypes.h>
24 #include <sys/socket.h>
25 #include <netdb.h>
26
27 #include <ev.h>
28 #include <pthread.h>
29
30 #define CLIENT_BUFFER_SIZE 32 * 1024
31
32 #define W_MALLOC(t, n) ((t*) calloc((n), sizeof(t)))
33 #define W_REALLOC(p, t, n) ((t*) realloc(p, (n) * sizeof(t)))
34 #define W_ERROR(f, ...) fprintf(stderr, "error: " f "\n", __VA_ARGS__)
35 #define UNUSED(x) ( (void)(x) )
36
```

```
37 struct Config;  
38 typedef struct Config Config;  
39 struct Stats;  
40 typedef struct Stats Stats;  
41 struct Worker;  
42 typedef struct Worker Worker;  
43 struct Client;  
44 typedef struct Client Client;  
45  
46 #include "client.h"  
47 #include "worker.h"  
48  
49  
50 struct Config {  
51     uint64_t req_count;  
52     uint8_t thread_count;  
53     uint16_t concur_count;  
54     uint8_t keep_alive;  
55  
56     char *request;  
57     uint32_t request_size;  
58     struct addrinfo *saddr;  
59 };  
60  
61 uint64_t str\_to\_uint64(char *str);  
62  
63 #endif
```

[One Level Up](#)

[Top Level](#)

src/weighttp.c - weighttp

Functions defined

- [forge_request](#)
- [main](#)
- [resolve_host](#)
- [show_help](#)
- [str_to_uint64](#)

Source code

```
1 /*
2  * weighttp - a lightweight and simple webserver benchmarking tool
3  *
4  * Author:
5  *   Copyright (c) 2009-2011 Thomas Porzelt
6  *
7  * License:
8  *   MIT, see COPYING file
9  */
10
11 #include "weighttp.h"
12
13 extern int optind, optopt; /* getopt */
14
15 static void show_help(void) {
16     printf("weighttp <options> <url>\n");
17     printf("  -n num   number of requests   (mandatory)\n");
18     printf("  -t num   threadcount          (default: 1)\n");
19     printf("  -c num   concurrent clients   (default: 1)\n");
20     printf("  -k       keep alive           (default: no)\n");
21     printf("  -6       use ipv6             (default: no)\n");
22     printf("  -H str   add header to request\n");
23     printf("  -h       show help and exit\n");
24     printf("  -v       show version and exit\n");
25     printf("example: weighttpd -n 10000 -c 10 -t 2 -k -H \"User-Agent: foo\" localhost/index.html\n\n");
26 }
27
28 static struct addrinfo *resolve_host(char *hostname, uint16_t port, uint8_t use_ipv6) {
29     int err;
30     char port_str[6];
31     struct addrinfo hints, *res, *res_first, *res_last;
32
33     memset(&hints, 0, sizeof(hints));
34     hints.ai_family = PF_UNSPEC;
35     hints.ai_socktype = SOCK_STREAM;
36
37     sprintf(port_str, "%d", port);
38
39     err = getaddrinfo(hostname, port_str, &hints, &res_first);
40
41     if (err) {
42         W_ERROR("could not resolve hostname: %s", hostname);
43         return NULL;
44     }
45
46     /* search for an ipv4 address, no ipv6 yet */
47     res_last = NULL;
48     for (res = res_first; res != NULL; res = res->ai_next) {
49         if (res->ai_family == AF_INET && !use_ipv6)
50             break;
51         else if (res->ai_family == AF_INET6 && use_ipv6)
52             break;
53     }
```

```

54     res_last = res;
55 }
56
57 if (!res) {
58     freeaddrinfo(res_first);
59     W_ERROR("could not resolve hostname: %s", hostname);
60     return NULL;
61 }
62
63 if (res != res_first) {
64     /* unlink from list and free rest */
65     res_last->ai_next = res->ai_next;
66     freeaddrinfo(res_first);
67     res->ai_next = NULL;
68 }
69
70 return res;
71 }
72
73 static char *forge_request(char *url, char keep_alive, char **host, uint16_t *port, char **headers, uint8_t
headers_num) {
74     char *c, *end;
75     char *req;
76     uint32_t len;
77     uint8_t i;
78     uint8_t have_user_agent, have_host;
79
80     *host = NULL;
81     *port = 0;
82
83     if (strncmp(url, "http://", 7) == 0)
84         url += 7;
85     else if (strncmp(url, "https://", 8) == 0) {
86         W_ERROR("%s", "no ssl support yet");
87         url += 8;
88         return NULL;
89     }
90
91     len = strlen(url);
92
93     if ((c = strchr(url, ':')) {
94         /* found ':' => host:port */
95         *host = W_MALLOC(char, c - url + 1);
96         memcpy(*host, url, c - url);
97         (*host)[c - url] = '\0';
98
99         if ((end = strchr(c+1, '/')) {
100             *end = '\0';
101             *port = atoi(c+1);
102             *end = '/';
103             url = end;
104         } else {
105             *port = atoi(c+1);
106             url += len;
107         }
108     } else {
109         *port = 80;
110
111         if ((c = strchr(url, '/')) {
112             *host = W_MALLOC(char, c - url + 1);
113             memcpy(*host, url, c - url);
114             (*host)[c - url] = '\0';
115             url = c;
116         } else {
117             *host = W_MALLOC(char, len + 1);
118             memcpy(*host, url, len);
119             (*host)[len] = '\0';
120             url += len;
121         }
122     }
123
124     if (*port == 0) {
125         W_ERROR("%s", "could not parse url");
126         free(*host);
127         return NULL;
128     }

```

```

129     if (*url == '\0')
130         url = "/";
131
132     // total request size
133     len = strlen("GET HTTP/1.1\r\nConnection: keep-alive\r\n\r\n") + 1;
134     len += strlen(url);
135
136     have_user_agent = 0;
137     have_host = 0;
138     for (i = 0; i < headers_num; i++) {
139         len += strlen(headers[i]) + strlen("\r\n");
140         if (strncmp(headers[i], "User-Agent: ", sizeof("User-Agent: ")-1) == 0)
141             have_user_agent = 1;
142         if (strncasecmp(headers[i], "Host:", sizeof("Host: ")-1) == 0)
143             have_host = 1;
144     }
145
146     if (!have_user_agent)
147         len += strlen("User-Agent: weighthttp/" VERSION "\r\n");
148
149     if (!have_host) {
150         len += strlen("Host: :65536\r\n")-1;
151         len += strlen(*host);
152     }
153
154     req = W_MALLOC(char, len);
155
156     strcpy(req, "GET ");
157     strcat(req, url);
158     strcat(req, " HTTP/1.1\r\n");
159
160     if (!have_host) {
161         strcat(req, "Host: ");
162         strcat(req, *host);
163         if (*port != 80)
164             sprintf(req + strlen(req), ":%"PRIu16, *port);
165         strcat(req, "\r\n");
166     }
167
168     if (!have_user_agent)
169         sprintf(req + strlen(req), "User-Agent: weighthttp/" VERSION "\r\n");
170
171     for (i = 0; i < headers_num; i++) {
172         strcat(req, headers[i]);
173         strcat(req, "\r\n");
174     }
175
176     if (keep_alive)
177         strcat(req, "Connection: keep-alive\r\n\r\n");
178     else
179         strcat(req, "Connection: close\r\n\r\n");
180
181     return req;
182 }
183
184
185 uint64_t str_to_uint64(char *str) {
186     uint64_t i;
187
188     for (i = 0; *str; str++) {
189         if (*str < '0' || *str > '9')
190             return UINT64_MAX;
191
192         i *= 10;
193         i += *str - '0';
194     }
195
196     return i;
197 }
198
199 int main(int argc, char *argv[]) {
200     Worker **workers;
201     pthread_t *threads;
202     int i;
203     char c;
204     int err;

```



```

205 struct ev_loop *loop;
206 ev_tstamp ts_start, ts_end;
207 Config config;
208 Worker *worker;
209 char *host;
210 uint16_t port;
211 uint8_t use_ipv6;
212 uint16_t rest_concur, rest_req;
213 Stats stats;
214 ev_tstamp duration;
215 int sec, millisec, microsec;
216 uint64_t rps;
217 uint64_t kbps;
218 char **headers;
219 uint8_t headers_num;
220
221 printf("weighttp - a lightweight and simple webserver benchmarking tool\n\n");
222
223 headers = NULL;
224 headers_num = 0;
225
226 /* default settings */
227 use_ipv6 = 0;
228 config.thread_count = 1;
229 config.concur_count = 1;
230 config.req_count = 0;
231 config.keep_alive = 0;
232
233 while ((c = getopt(argc, argv, "hv6kn:t:c:H:")) != -1) {
234     switch (c) {
235         case 'h':
236             show_help();
237             return 0;
238         case 'v':
239             printf("version:      " VERSION "\n");
240             printf("build-date: " __DATE__ " " __TIME__ "\n\n");
241             return 0;
242         case '6':
243             use_ipv6 = 1;
244             break;
245         case 'k':
246             config.keep_alive = 1;
247             break;
248         case 'n':
249             config.req_count = str_to_uint64(optarg);
250             break;
251         case 't':
252             config.thread_count = atoi(optarg);
253             break;
254         case 'c':
255             config.concur_count = atoi(optarg);
256             break;
257         case 'H':
258             headers = W_REALLOC(headers, char*, headers_num+1);
259             headers[headers_num] = optarg;
260             headers_num++;
261             break;
262         case '?':
263             W_ERROR("unkown option: -%c", optopt);
264             show_help();
265             return 1;
266     }
267 }
268
269 if ((argc - optind) < 1) {
270     W_ERROR("%s", "missing url argument\n");
271     show_help();
272     return 1;
273 } else if ((argc - optind) > 1) {
274     W_ERROR("%s", "too many arguments\n");
275     show_help();
276     return 1;
277 }
278
279 /* check for sane arguments */
280 if (!config.thread_count) {

```

```

281     W_ERROR("%s", "thread count has to be > 0\n");
282     show_help();
283     return 1;
284 }
285 if (!config.concur_count) {
286     W_ERROR("%s", "number of concurrent clients has to be > 0\n");
287     show_help();
288     return 1;
289 }
290 if (!config.req_count) {
291     W_ERROR("%s", "number of requests has to be > 0\n");
292     show_help();
293     return 1;
294 }
295 if (config.req_count == UINT64_MAX || config.thread_count > config.req_count || config.thread_count >
config.concur_count || config.concur_count > config.req_count) {
296     W_ERROR("%s", "insane arguments\n");
297     show_help();
298     return 1;
299 }
300
301 loop = ev_default_loop(0);
302 if (!loop) {
303     W_ERROR("%s", "could not initialize libev\n");
304     return 2;
305 }
306 }
307
308 if (NULL == (config.request = forge_request(argv[optind], config.keep_alive, &host, &port, headers,
headers_num))) {
309     return 1;
310 }
311
312 config.request_size = strlen(config.request);
313 //printf("Request (%d):\n=====\n%s=====\n", config.request_size, config.request);
314 //printf("host: '%s', port: %d\n", host, port);
315
316 /* resolve hostname */
317 if(!(config.saddr = resolve_host(host, port, use_ipv6))) {
318     return 1;
319 }
320
321 /* spawn threads */
322 threads = W_MALLOC(pthread_t, config.thread_count);
323 workers = W_MALLOC(Worker*, config.thread_count);
324
325 rest_concur = config.concur_count % config.thread_count;
326 rest_req = config.req_count % config.thread_count;
327
328 printf("starting benchmark...\n");
329
330 memset(&stats, 0, sizeof(stats));
331 ts_start = ev_time();
332
333 for (i = 0; i < config.thread_count; i++) {
334     uint64_t reqs = config.req_count / config.thread_count;
335     uint16_t concur = config.concur_count / config.thread_count;
336     uint64_t diff;
337
338     if (rest_concur) {
339         diff = (i == config.thread_count) ? rest_concur : (rest_concur / config.thread_count);
340         diff = diff ? diff : 1;
341         concur += diff;
342         rest_concur -= diff;
343     }
344
345     if (rest_req) {
346         diff = (i == config.thread_count) ? rest_req : (rest_req / config.thread_count);
347         diff = diff ? diff : 1;
348         reqs += diff;
349         rest_req -= diff;
350     }
351     printf("spawning thread #%d: %"PRIu16" concurrent requests, %"PRIu64" total requests\n", i+1, concur,
reqs);
352     workers[i] = worker_new(i+1, &config, concur, reqs);
353

```

```

354     if (!(workers[i])) {
355         W_ERROR("%s", "failed to allocate worker or client");
356         return 1;
357     }
358
359     err = pthread_create(&threads[i], NULL, worker_thread, (void*)workers[i]);
360
361     if (err != 0) {
362         W_ERROR("failed spawning thread (%d)", err);
363         return 2;
364     }
365 }
366
367 for (i = 0; i < config.thread_count; i++) {
368     err = pthread_join(threads[i], NULL);
369     worker = workers[i];
370
371     if (err != 0) {
372         W_ERROR("failed joining thread (%d)", err);
373         return 3;
374     }
375
376     stats.req_started += worker->stats.req_started;
377     stats.req_done += worker->stats.req_done;
378     stats.req_success += worker->stats.req_success;
379     stats.req_failed += worker->stats.req_failed;
380     stats.bytes_total += worker->stats.bytes_total;
381     stats.bytes_body += worker->stats.bytes_body;
382     stats.req_2xx += worker->stats.req_2xx;
383     stats.req_3xx += worker->stats.req_3xx;
384     stats.req_4xx += worker->stats.req_4xx;
385     stats.req_5xx += worker->stats.req_5xx;
386
387     worker_free(worker);
388 }
389
390 ts_end = ev_time();
391 duration = ts_end - ts_start;
392 sec = duration;
393 duration -= sec;
394 duration = duration * 1000;
395 millisec = duration;
396 duration -= millisec;
397 microsec = duration * 1000;
398 rps = stats.req_done / (ts_end - ts_start);
399 kbps = stats.bytes_total / (ts_end - ts_start) / 1024;
400 printf("\nfinished in %d sec, %d millisec and %d microsec, %"PRIu64" req/s, %"PRIu64" kbyte/s\n", sec,
millisec, microsec, rps, kbps);
401 printf("requests: %"PRIu64" total, %"PRIu64" started, %"PRIu64" done, %"PRIu64" succeeded, %"PRIu64"
failed, %"PRIu64" errored\n",
402     config.req_count, stats.req_started, stats.req_done, stats.req_success, stats.req_failed,
stats.req_error
403 );
404 printf("status codes: %"PRIu64" 2xx, %"PRIu64" 3xx, %"PRIu64" 4xx, %"PRIu64" 5xx\n",
405     stats.req_2xx, stats.req_3xx, stats.req_4xx, stats.req_5xx
406 );
407 printf("traffic: %"PRIu64" bytes total, %"PRIu64" bytes http, %"PRIu64" bytes data\n",
408     stats.bytes_total, stats.bytes_total - stats.bytes_body, stats.bytes_body
409 );
410
411 ev_default_destroy();
412
413 free(threads);
414 free(workers);
415 free(config.request);
416 free(host);
417 free(headers);
418 freeaddrinfo(config.saddr);
419
420 return 0;
421 }

```

src/worker.c - weighttp

Functions defined

- [worker_free](#)
- [worker_new](#)
- [worker_thread](#)

Source code

```
1  /*
2  * weighttp - a lightweight and simple webserver benchmarking tool
3  *
4  * Author:
5  *   Copyright (c) 2009-2011 Thomas Porzelt
6  *
7  * License:
8  *   MIT, see COPYING file
9  */
10
11 #include "weighttp.h"
12
13 Worker *worker_new(uint8_t id, Config *config, uint16_t num_clients, uint64_t num_requests) {
14     Worker *worker;
15     uint16_t i;
16
17     worker = W_MALLOC(Worker, 1);
18     worker->id = id;
19     worker->loop = ev_loop_new(0);
20     ev_ref(worker->loop);
21     worker->config = config;
22     worker->num_clients = num_clients;
23     worker->stats.req_todo = num_requests;
24     worker->progress_interval = num_requests / 10;
25
26     if (worker->progress_interval == 0)
27         worker->progress_interval = 1;
28
29     worker->clients = W_MALLOC(Client*, num_clients);
30
31     for (i = 0; i < num_clients; i++) {
32         if (NULL == (worker->clients[i] = client_new(worker)))
33             return NULL;
34     }
35
36     return worker;
37 }
38
39 void worker_free(Worker *worker) {
40     uint16_t i;
41
42     for (i = 0; i < worker->num_clients; i++)
43         client_free(worker->clients[i]);
44
45     free(worker->clients);
46     free(worker);
47 }
48
49 void *worker_thread(void* arg) {
50     uint16_t i;
51     Worker *worker = (Worker*)arg;
52
53     /* start all clients */
54     for (i = 0; i < worker->num_clients; i++) {
55         if (worker->stats.req_started < worker->stats.req_todo)
56             client_state_machine(worker->clients[i]);
57     }
58 }
```

```
59     ev_loop(worker->loop, 0);
60
61     ev_loop_destroy(worker->loop);
62
63     return NULL;
64 }
```

[One Level Up](#)

[Top Level](#)