Alexandria University

Ziyad Ahmed Elbanna -26

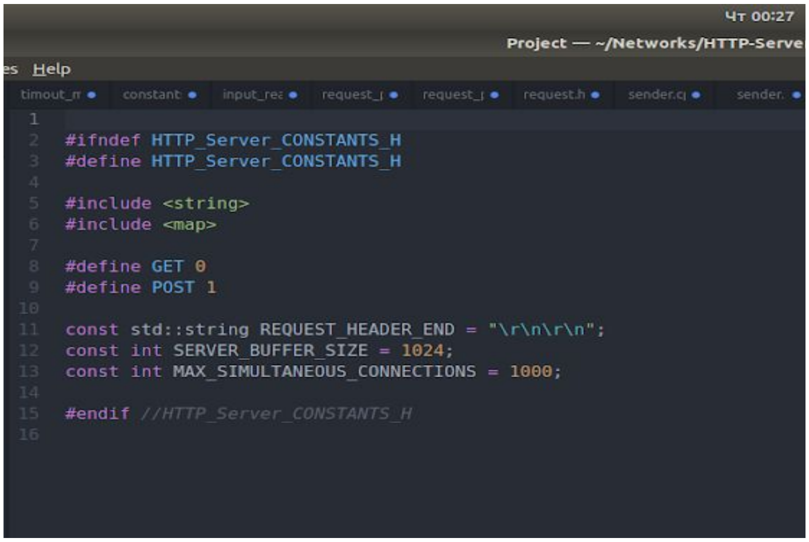
Assignment 1

Introduction to Socket Programming

Part 1: Server side

Code Organization

1. Constants are gathered in server_constants.h



```
#ifndef HTTP_Server_CONSTANTS_H
#define HTTP_Server_CONSTANTS_H

#include <string>
#include <map>

#define GET 0
#define POST 1

const std::string REQUEST_HEADER_END = "\r\n\r\n";
const int SERVER_BUFFER_SIZE = 1024;
const int MAX_SIMULTANEOUS_CONNECTIONS = 1000;

#endif //HTTP_Server_CONSTANTS_H
```

2. Request_handler.cpp

➔ Responsible for dealing with client's reques

3. Request_handler.cpp

➔ Build the head map for the accepted requests.

4. Socket_manager.cpp

→ Handles creation of server's socket file descriptor for each accepted client.

5. Timeout_manager.cpp

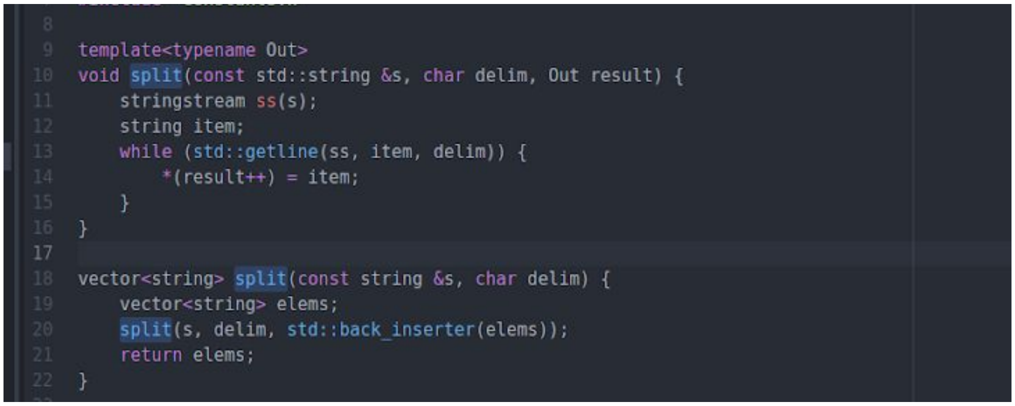
→ Update timeout for open sockets depending on the percentage of the active clients to maximum number of allowed connections.

Major functions

1. Get_socket_fd

→ Creates a socket file descriptor for the server and binds it with a specific IP address.

2. Split

→ 

```
template<typename Out>
void split(const std::string &s, char delim, Out result) {
    stringstream ss(s);
    string item;
    while (std::getline(ss, item, delim)) {
        *(result++) = item;
    }
}

vector<string> split(const string &s, char delim) {
    vector<string> elems;
    split(s, delim, std::back_inserter(elems));
    return elems;
}
```

Splitting the request by delimeters

3. handle_request

→ Handling new requests coming to server, ensuring persistent connections for accepting multiple requests through same connection.

4. Get_response_header

→ Helper method for handling request, Process first part of the request after reading the header.

5. Update_timeout

→ Update the timeout for all opened sockets, by using an equation The function is synchronized through mutex.

$$(3 * \frac{MAX\ SIMULTANEOUS\ CONNECTIONS}{number\ of\ open\ sockets + 1}) + 1$$

Major datastructure

1. Struct request for Request Data
2. Struct server for Server Info

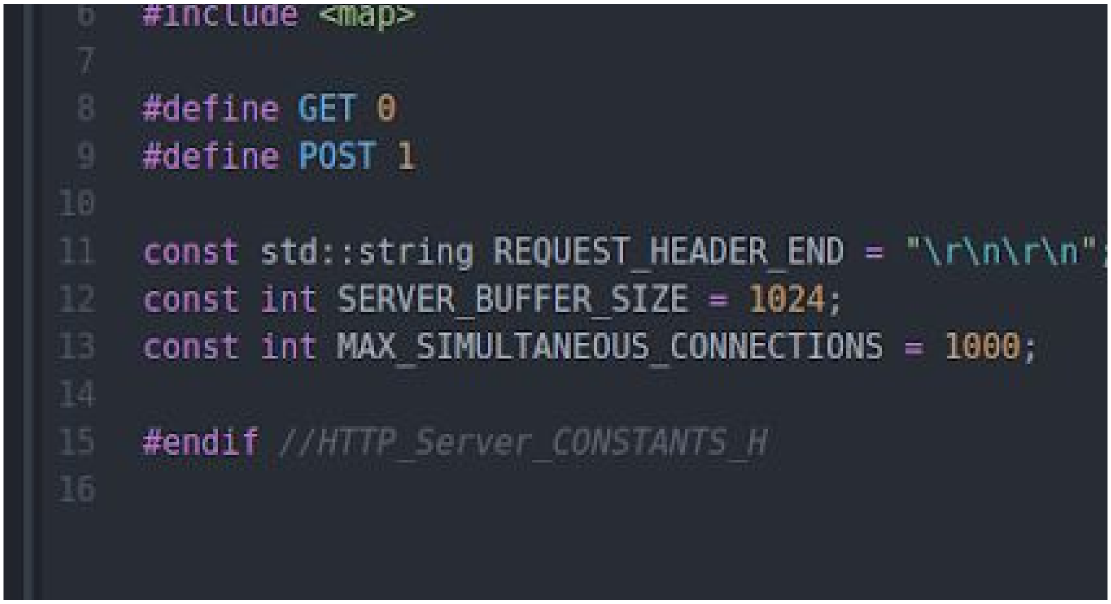
SPECIFICATIONS

- I chose to make the server Multi-threaded not Multi-process as threads are lighter than processes and share the same address space, also passing data doesn't need message passing. Each Client will have a serving thread with a limit on the number of concurrent active threads.

Part 2: Client side

● Code Organization:

1. Constants are gathered in constants.h



```
#include <map>

#define GET 0
#define POST 1

const std::string REQUEST_HEADER_END = "\r\n\r\n";
const int SERVER_BUFFER_SIZE = 1024;
const int MAX_SIMULTANEOUS_CONNECTIONS = 1000;

#endif //HTTP_Server_CONSTANTS_H

```

2. Input_reader.cpp

➔ Responsible for opening and reading input file.

3. Request_parser.cpp

➔ Parses the request to obtain file name, port number, hostname, request type.

4. Sender.cpp

➔ Responsible with dealing with server whether in post or get request.

5. Sockets_manager.cpp

➔ Connects a client's socket file descriptor to server with the required host name and port number.

● Major Functions:

1. Read_requests_from_file

   ➔ Opens the input file to start reading the requests.

2. Get_requests_vector

   ➔ Reads input file line by line and returns a vector<vector<request>>.

3. Get_key

   ➔ Creates a request key in the formate HostName#PortNumber.

4. Process_requests

   ➔ Process each request using the socket fd created.

5. Get_socket_fd

   ➔ Returns a client's socket file descriptor that is connected to a desired server.

6. Split

   ➔ Splits the request by a delimiter.

7. Parse_request

   ➔ Extracts file name, port number, hostname, request type out of a

   Request.

- **Data Structures:**

1. Struct request for Request Data
2. vector<vector<request>>

   ➔ Contains the requests read from the input file and processed later.

3. Bonus part

a. Client part testing

⟶ Client was tested using [Henry's Post Test Server V2] which is a service for developers testing clients that POST and GET things over HTTP.

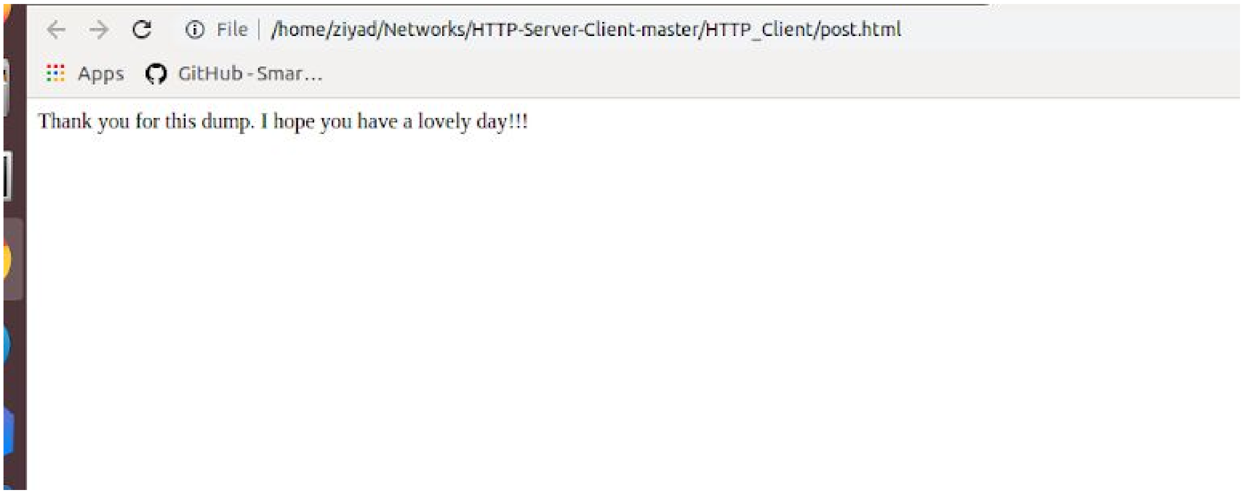
⟶ The input file passed was in the form:

GET /t/jti0q-1542455122/post ptsv2.com

GET /t/jti0q-1542455122/post ptsv2.com

GET /t/jti0q-1542455122/post ptsv2.com

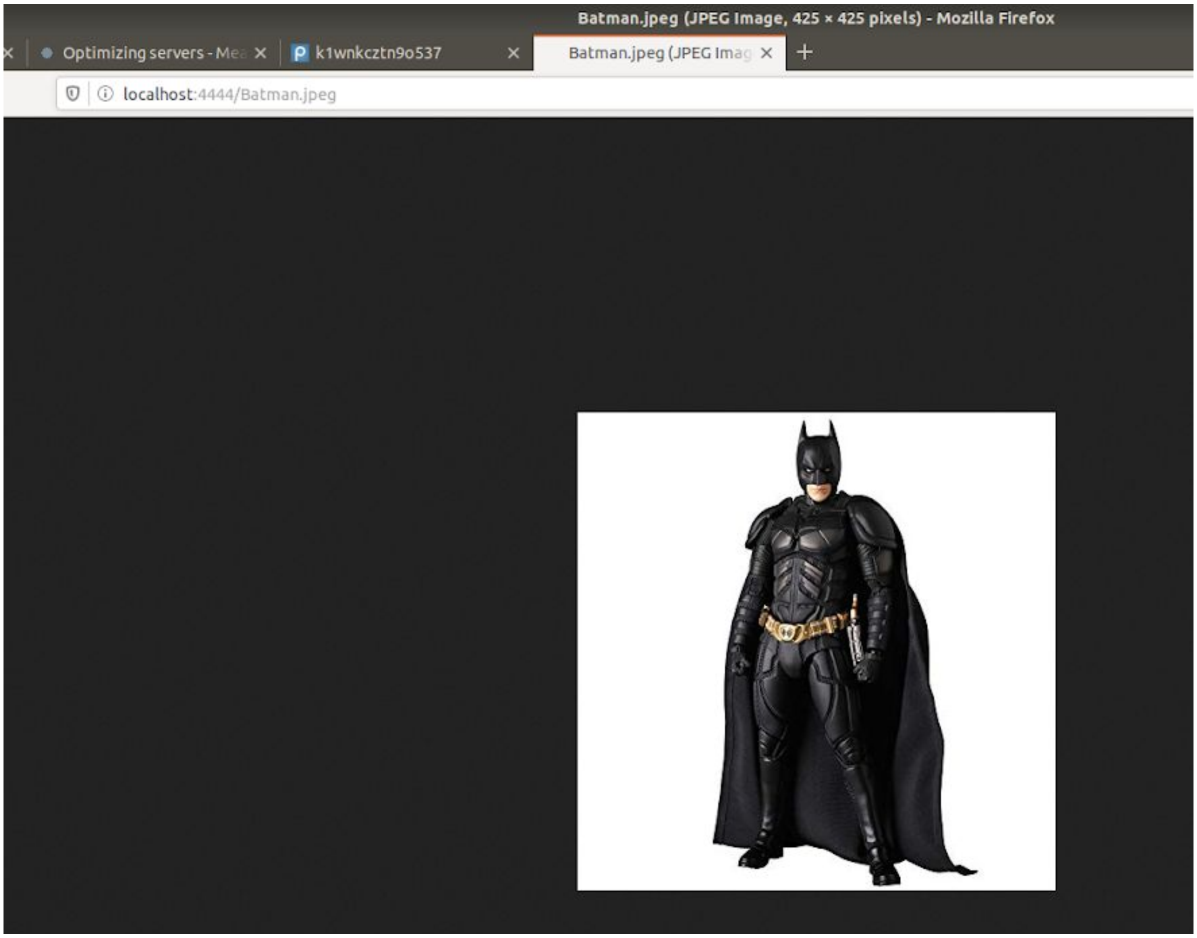
⟶ The result was


File
/home/ziyad/Networks/HTTP-Server-Client-master/HTTP_Client/post.html
Apps
GitHub - Smar...

Thank you for this dump. I hope you have a lovely day!!!

b. Testing the server part

Server was tested using firefox

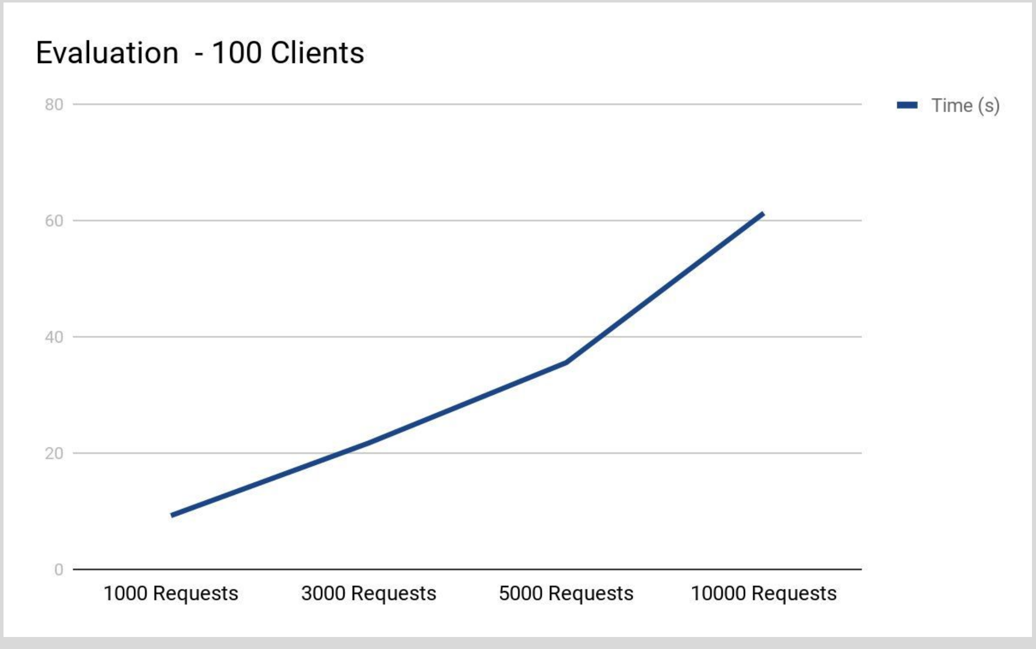

Batman.jpeg (JPEG Image, 425 × 425 pixels) - Mozilla Firefox
Optimizing servers - Mea
k1wnkcztn9o537
Batman.jpeg (JPEG Imag
localhost:4444/Batman.jpeg

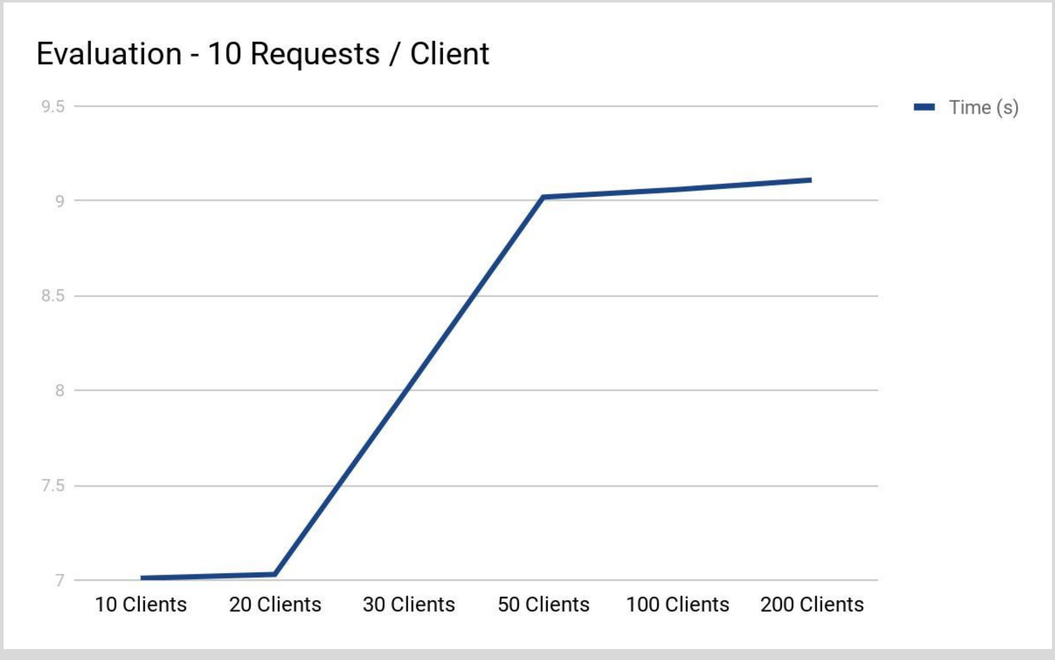
c. Performance Evaluation

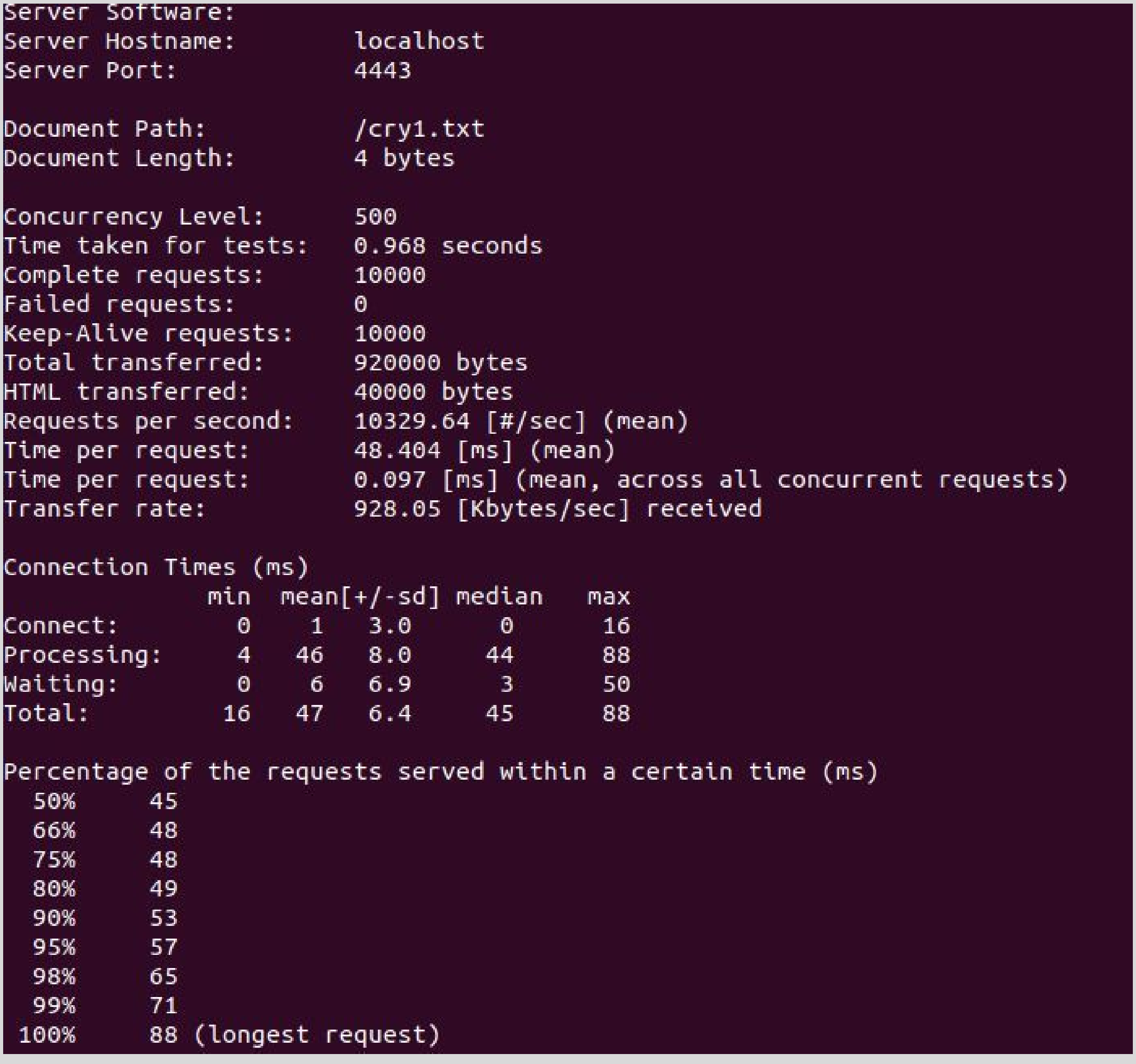
This command was typed in the Apache benchmark tool:

```
siege -u  127.0.0.1:4444/batman.jpeg -d1 -r100 -c10
```

Reps = 100, concurrent users = 10 and delay = 1







```
Server Software:
Server Hostname:        localhost
Server Port:            4443

Document Path:          /cry1.txt
Document Length:        4 bytes

Concurrency Level:      500
Time taken for tests:   0.968 seconds
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    10000
Total transferred:      920000 bytes
HTML transferred:       40000 bytes
Requests per second:    10329.64 [#/sec] (mean)
Time per request:       48.404 [ms] (mean)
Time per request:       0.097 [ms] (mean, across all concurrent requests)
Transfer rate:          928.05 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   3.0      0      16
Processing:     4   46   8.0     44      88
Waiting:        0    6   6.9      3      50
Total:         16   47   6.4     45      88

Percentage of the requests served within a certain time (ms)
  50%     45
  66%     48
  75%     48
  80%     49
  90%     53
  95%     57
  98%     65
  99%     71
 100%     88 (longest request)
```