Laporan Ujian Akhir Semester Pemrograman Jaringan

Kelompok 6

Oleh :

1.	Wasilatul Dewi Ningrum	(05111740000004)
2.	Hisam Widi Prayoga	(05111740000026)
3.	Putri Endah Puspitasari	(05111740000039)
4.	Arini Puspitasari	(05111740000040)

Kelas : Pemrograman Jaringan - C

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

SURABAYA

2020

I. PENJELASAN KODINGAN

1.1 Penjelasan kodingan async_server.py dan http.py

```
class Server(asyncore.dispatcher):
    def __init__(self,portnumber):
        asyncore.dispatcher.__init__(self)
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.set_reuse_addr()
        self.bind(('',portnumber))
        self.listen(5)
        logging.warning("running on port {}" . format(portnumber))

    def handle_accept(self):
        pair = self.accept()
        if pair is not None:
            sock, addr = pair
            logging.warning("connection from {}" . format(repr(addr)))
            handler = ProcessTheClient(sock)
```

Pada class Server terdapat dua fungsi yaitu yang pertama **def__init__** dan yang kedua adalah **def handle_accept**. Pada fungsi handle_accept ini ada kondisi apabila terdapat koneksi maka akan memanggil socket **ProcessTheClient**.

ProcessTheClient ini merupakan salah satu fungsi yang ada di dalam async_server.py.

```
class ProcessTheClient(asyncore.dispatcher_with_send):
    def handle_read(self):
        global rcv
        data = self.recv(1024)
        if data:
            d = data.decode()
            rcv = rcv + d
            if rcv[-2:] == '\r\n':
                hasil = httpserver.proses(rcv)
                #hasil sudah dalam bentuk bytes
                hasil = hasil + "\r\n\r\n".encode()
                self.send(hasil) #hasil sudah dalam bentuk bytes, kirimk
an balik ke client
                rcv = ""
                self.close()
        self.close()
```

Di class ProcessTheClient ini terdapat fungsi **def handle_read** yang berfungi untuk membaca request dari client. Jika terdapat request, maka request itu selanjutnya akan di proses di httpserver seperti yang terlihat pada salah satu line di potongan kodingan di atas.

Kemudian kita dapat melihat isi dari httpserver itu yaitu melalui file http.py. Mari kita lihat isi dari file http.py.

```
def proses(self,data):

        requests = data.split("\r\n")
        #print(requests)

        baris = requests[0]
        #print(baris)

        all_headers = [n for n in requests[1:] if n!='']

        j = baris.split(" ")
        try:
            method=j[0].upper().strip()
            if (method=='GET'):
                object_address = j[1].strip()
                return self.http_get(object_address, all_headers)
            if (method=='POST'):
                object_address = j[1].strip()
                return self.http_post(object_address, all_headers)
            else:
                return self.response(400,'Bad Request','',{})
        except IndexError:
            return self.response(400,'Bad Request','',{})
```

Fungsi **def proses** seperti yang tercantum diatas secara garis besar digunakan untuk menentukan request yang ada dengan method get atau method post. Selanjutnya terdapat fungsi **http_get** pada file http.py yang cuplikan kodingan dan penjelasan nya akan dijelaskan di bawah ini.

```
def http_get(self,object_address,headers):
        files = glob('./*')
        thedir='.'
        if thedir+object_address not in files:
            return self.response(404,'Not Found','',{})
        fp = open(thedir+object_address,'rb') #rb => artinya adalah read
 dalam bentuk binary
        #harus membaca dalam bentuk byte dan BINARY
        isi = fp.read()

        fext = os.path.splitext(thedir+object_address)[1]
        content_type = self.types[fext]

        headers={}
        headers['Content-type']=content_type
        return self.response(200,'OK',isi,headers)
```

Fungsi **http_get** ini digunakan untuk mencari apa yang di get misalnya yang di get adalah page.html, maka akan mencari di dalam folder ada atau tidak page.html di dalamnya. Selain itu **http_get** ini juga digunakan untuk memproses request dengan method get. Selanjutnya terdapat fungsi response pada file http.py yang cuplikan kodingan dan penjelasan nya akan dijelaskan di bawah ini.

```
def response(self,kode=404,message='Not Found',messagebody=bytes(),heade
rs={}):
        tanggal = datetime.now().strftime('%c')
        resp=[]
        resp.append("HTTP/1.0 {} {}\r\n" . format(kode,message))
        resp.append("Date: {}\r\n" . format(tanggal))
        resp.append("Connection: close\r\n")
        resp.append("Server: myserver/1.0\r\n")
        resp.append("Content-Length: {}\r\n" . format(len(messagebody)))
        for kk in headers:
            resp.append("{}:{}\r\n" . format(kk,headers[kk]))
        resp.append("\r\n")

        response_headers=''
        for i in resp:
            response_headers="{}{}" . format(response_headers,i)
        #menggabungkan resp menjadi satu string dan menggabungkan dengan
 messagebody yang berupa bytes
        #response harus berupa bytes
        #message body harus diubah dulu menjadi bytes
        if (type(messagebody) is not bytes):
            messagebody = messagebody.encode()

        response = response_headers.encode() + messagebody
        #response adalah bytes
        return response
```

Untuk fungsi response diatas menggambarkan proses menyiapkan format respon yang kemudian akan dikirim ke client.

Proses terakhir yaitu mengembalikan kembali kepada client melalui syntax *'self.send(hasil)'* yang terdapat pada file async_server.py.

2.1 Penjelasan mengenai kodingan lb.py

```
def __init__(self):

        self.servers=[]
        self.servers.append(('127.0.0.1',9002))
        self.servers.append(('127.0.0.1',9003))
        self.servers.append(('127.0.0.1',9004))
        self.servers.append(('127.0.0.1',9005))
        self.most_port = 9005
        self.most_treshold = 100
        self.client_num = 0
        self.current=0
```

Fungsi **__init__** yang terdapat di dalam class **BackendList** berfungsi untuk mencatat suatu list worker atau bisa disebut juga dengan list server dalam program ini.

```
def getserver(self,client_num):

        s = self.servers[self.current]
        self.current=self.current+1
        if (self.current>=len(self.servers)):
            self.current=0
        self.client_num = client_num
        return s
```

Potongan kodingan di atas merupakan fungsi **getserver** yang masih di dalam class BackendList yang berfungsi untuk menentukan server mana yang akan digunakan untuk memproses suatu request yang diberikan oleh client.

```
def checkConnection(self):

        if self.client_num > self.most_treshold:
            logging.warning("starting server")
            cmd = """ python3 async_server.py %d &""" % (self.most_port+
1)
            logging.warning(cmd)
            res = os.system(cmd)
            if res:
                logging.warning("failed to start new server at {}" . for
mat(self.most_port+1))
                return
            time.sleep(.5)
            logging.warning("new server is starting at port {}" . format
(self.most_port))
            self.addNewServer(self.most_port+1,self.most_treshold+50)
```

Fungsi **checkConnection** yang juga masih termasuk di dalam class BackendList digunakan untuk mengecek jumlah client yang mengakses Load Balancer ini, apakah lebih banyak dari threshold yang ditentukan atau belum.

```
def addNewServer(self,new_port, new_treshold):

        self.servers.append(('127.0.0.1',new_port))
        self.most_port = new_port
        self.most_treshold = new_treshold
```

Fungsi **addNewServer** yang terdapat di dalam class **BackendList** berfungsi untuk menambah list server yang sedang aktif. Setelah mengisikan server ke dalam new_port maka, server yang baru telah berhasil ditambahkan ke dalam list server yang berada di dalam fungsi **__init** di atas.

```
class Backend(asyncore.dispatcher_with_send):

    def __init__(self,targetaddress):
        asyncore.dispatcher_with_send.__init__(self)
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connect(targetaddress)
        self.connection = self

    def handle_read(self):
        try:
            self.client_socket.send(self.recv(8192))
        except:
            pass
    def handle_close(self):
        try:
            self.close()
            self.client_socket.close()
        except:
            pass
```

Class **Backend** yang memiliki fungsi **__init__**, **handle_read**, dan **handle_close** digunakan untuk mengirimkan suatu respon yang diberikan oleh server ke client yang bersangkutan.

```
class ProcessTheClient(asyncore.dispatcher):

    def handle_read(self):
        data = self.recv(8192)
        if data:
            self.backend.client_socket = self
            self.backend.send(data)
    def handle_close(self):
        self.close()
```

Sedangkan untuk class **ProcessTheClient** ini sendiri memiliki fungsi yaitu untuk meneruskan request dari client ke server.

```
class Server(asyncore.dispatcher):

    def __init__(self,portnumber):
        asyncore.dispatcher.__init__(self)
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.set_reuse_addr()
        self.bind(('',portnumber))
        self.listen(5)
        self.bservers = BackendList()
        self.timer = ThreadCheck(self.bservers)
        self.timer.start()
```

Kodingan di atas merupakan class **Server** yang berfungsi untuk menerima request dari client dan meneruskan request tersebut ke server yang telah dipilih sebelumnya.

```
bs = self.bservers.getserver(len(asyncore.socket_map))
            logging.warning("koneksi dari {} diteruskan ke {}" . format(
addr, bs))
            backend = Backend(bs)
```

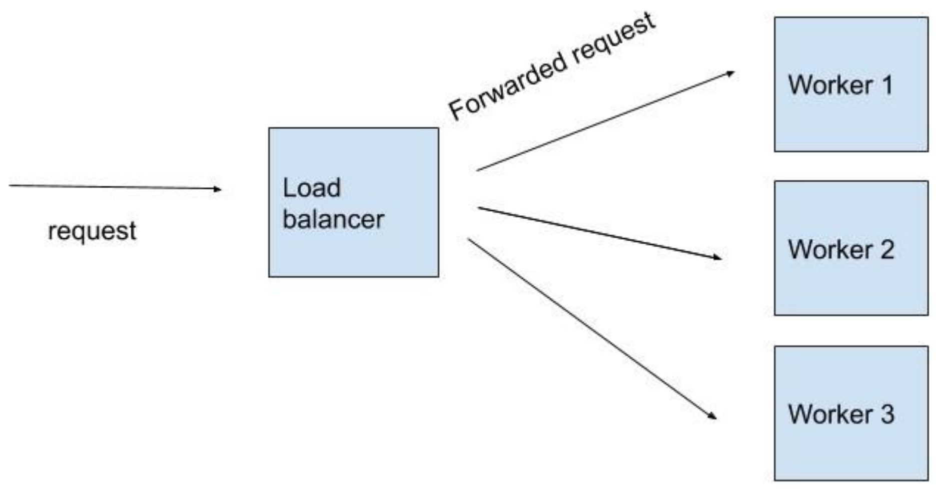
Potongan kodingan di atas merupakan bagian dari fungsi **handle_accept** yang terdapat di dalam class berguna untuk menentukan server mana yang akan digunakan untuk meneruskan request dari client.

```
handler = ProcessTheClient(sock)
            handler.backend = backend
```

Potongan kodingan di atas merupakan bagian dari fungsi handle_accept yang terdapat di dalam class Server berguna untuk mendapatkan handler dan juga socket dari client.

II. MODEL PROCESSING SERVER

Program ini mengimplementasikan ilustrasi berikut :

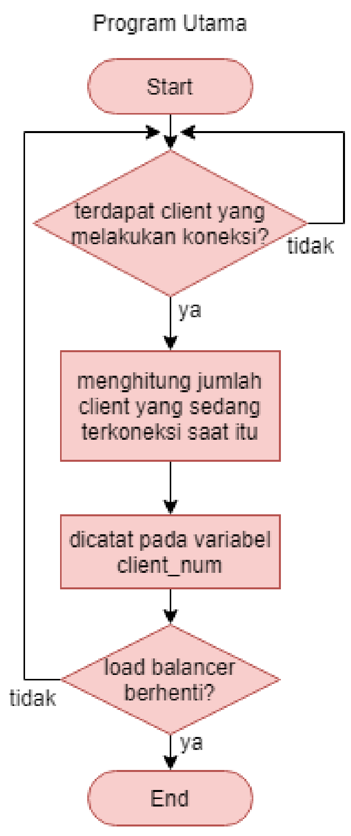


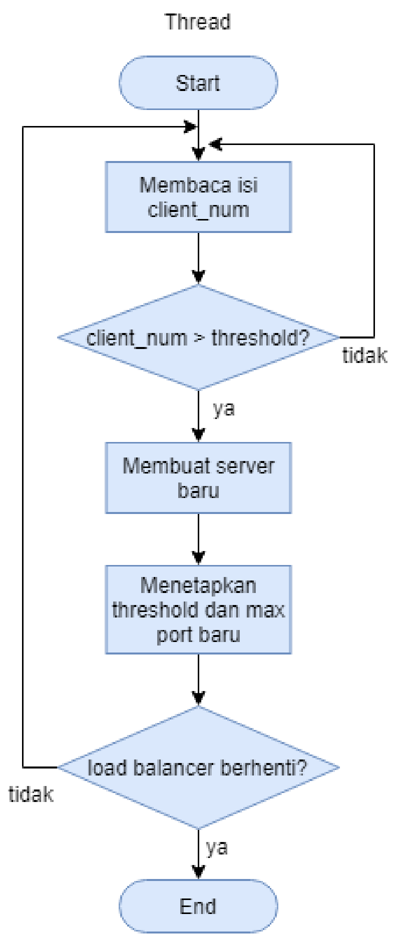
Dalam ilustrasi tersebut, setiap request yang masuk akan menuju load balancer server. Kemudian load balancer tersebut akan meneruskan request ke sebuah worker yang dipilih untuk diproses. Worker merupakan web server. Kemudian worker akan mengirimkan response ke load balancer untuk kemudian diteruskan ke client.

Web server maupun load balancer server diimplementasikan dengan model asynchronous karena dengan asynchronous processing alokasi memori dan CPU akan lebih efisien.

III. MEKANISME PENAMBAHAN WORKER

Load balancer adalah server yang bertugas untuk meneruskan request client ke sebuah worker. Worker adalah asynchronous web server yang bertugas untuk melayani request dari client. Pada awalnya terdapat tiga worker yang bekerja. Worker dapat ditambah sesuai dengan kebutuhan. Pada program ini, worker akan otomatis ditambahkan ketika jumlah client yang meminta layanan ke server melebihi threshold yang ditentukan. Ketika terdapat sebuah client yang melakukan koneksi dengan server, server load balancer akan menghitung jumlah client yang sedang melakukan koneksi saat itu. Lalu terdapat thread yang melakukan pengecekan jumlah client tiap 0.1 detik. Jika jumlah client yang melakukan koneksi lebih dari threshold yang ditentukan, maka server load balancer akan mengeksekusi pembuatan server baru dan mencatatnya pada BackendList agar dapat dipilih untuk digunakan. Alur tersebut dapat dilihat pada gambar berikut :


Program Utama
Start
terdapat client yang melakukan koneksi?
tidak
ya
menghitung jumlah client yang sedang terkoneksi saat itu
dicatat pada variabel client_num
load balancer berhenti?
tidak
ya
End



Sedangkan implementasi pada **lb.py** adalah sebagai berikut:

```
    def handle_accept(self):
        pair = self.accept()
        if pair is not None:
            sock, addr = pair
            # logging.warning("connection from {}" . format(repr(addr)))

            #menentukan ke server mana request akan diteruskan
            bs = self.bservers.getserver(len(asyncore.socket_map))
```

Potongan kode tersebut menunjukkan bahwa ketika terdapat sebuah client yang melakukan koneksi, load balancer akan melakukan pemilihan asynchronous server untuk melayani request tersebut. Saat memanggil fungsi getserver, disertakan pula fungsi untuk menghitung jumlah client yang terkoneksi dengan load balancer.

```
    def getserver(self,client_num):
        s = self.servers[self.current]
        self.current=self.current+1
        if (self.current>=len(self.servers)):
            self.current=0
        self.client_num = client_num
```

Jumlah client yang terkoneksi dengan load balancer disimpan dalam sebuah variabel.

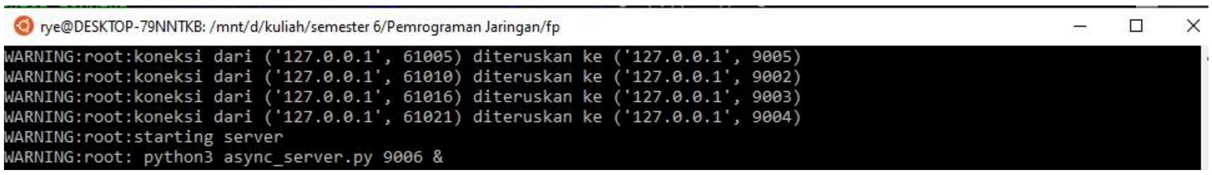
```
    def checkConnection(self):
        if self.client_num > self.most_treshold:
            logging.warning("starting server")
            cmd = """ python3 async_server.py %d &""" % (self.most_port+1)
            logging.warning(cmd)
            res = os.system(cmd)
            if res:
                logging.warning("failed to start new server at {}" . format(self.
most_port+1))
                return
            time.sleep(.5)
            logging.warning("new server is starting at port {}" . format(self.mos
t_port))
            self.addNewServer(self.most_port+1,self.most_treshold+50)
```

Fungsi checkConnection digunakan untuk melakukan pengecekan apakah jumlah client yang terkoneksi dengan load balancer melebihi threshold. Jika jumlah client yang terkoneksi dengan load balancer melebihi threshold, maka server baru akan dijalankan dengan mengeksekusi perintah “python3 async_server.py [port] &”. Proses dijeda selama 0.5 detik untuk memberi waktu memastikan server baru telah berjalan dan siap digunakan sehingga tidak error saat diakses. Lalu fungsi addNewServer dipanggil.

```
    def addNewServer(self,new_port, new_treshold):
        self.servers.append(('127.0.0.1',new_port))
        self.most_port = new_port
        self.most_treshold = new_treshold
```

Fungsi addNewServer digunakan untuk menambahkan informasi server (IP dan port) pada array servers di BackendList. Informasi tersebut perlu ditambahkan agar load balancer dapat memilih server tersebut untuk melayani request client.

Ketika program dijalankan, awalnya hanya memiliki tiga web server. Ketika terdapat request dalam jumlah besar, load balancer akan menjalankan server baru.


rye@DESKTOP-79NNTKB: /mnt/d/kuliah/semester 6/Pemrograman Jaringan/fp

```
WARNING:root:koneksi dari ('127.0.0.1', 61005) diteruskan ke ('127.0.0.1', 9005)
WARNING:root:koneksi dari ('127.0.0.1', 61010) diteruskan ke ('127.0.0.1', 9002)
WARNING:root:koneksi dari ('127.0.0.1', 61016) diteruskan ke ('127.0.0.1', 9003)
WARNING:root:koneksi dari ('127.0.0.1', 61021) diteruskan ke ('127.0.0.1', 9004)
WARNING:root:starting server
WARNING:root: python3 async_server.py 9006 &
```

IV. TABEL EKSPERIMEN

Performance Test dengan Parameter :

- Load Balancer tanpa Penambahan Otomatis

Jumlah Request	Konkurensi
1000	100, 300, 500, 800, 1000

- Load Balancer dengan penambahan otomatis

Jumlah Request	Konkurensi
1000	100, 300, 500, 800, 1000

Load Balancer tanpa penambahan otomatis (hanya 3 worker)

No. Test	Concurrency Level	Time Taken for Test [seconds]	Complete Request	Failed Request	Total Transferred [bytes]	Request per Second [#/sec]	Time Per Request [ms]	Transfer Rate [Kbytes/sec]
1	100	1.143	1000	0	226000	874.96	114.291	193.11
2	300	0.661	1000	0	226000	1513.38	198.232	334.01
3	500	1.409	1000	0	226000	709.91	704.316	156.68
4	800	1.510	1000	245	30500	662.12	1208.232	19.72
5	1000	1.371	1000	326	39772	729.52	1370.770	28.33

Load Balancer dengan penambahan otomatis (diawali 3 worker dan mengalami penambahan)

No. Test	Concurrency Level	Time Taken for Test [seconds]	Complete Request	Failed Request	Total Transferred [bytes]	Request per Second [#/sec]	Time Per Request [ms]	Transfer Rate [Kbytes/sec]
1	100	0.345	1000	0	226000	2899.77	34.486	639.99
2	300	1.104	1000	0	226000	906.11	331.087	199.98

3	500	1.067	1000	0	226000	937.17	533.521	206.84
4	800	1.008	1000	202	26718	991.95	806.490	25.88
5	1000	1.302	1000	165	20130	768.08	1301.952	15.10

V. KESIMPULAN

Penambahan worker yang berupa asynchronous web server dapat diimplementasikan menggunakan load balancer. Hal tersebut memberi dampak pada waktu yang stabil dan kecepatan transfer yang stabil juga, meskipun concurrency bertambah, dampak lainnya juga membuat failed request menjadi lebih sedikit.

VI. SOURCE CODE

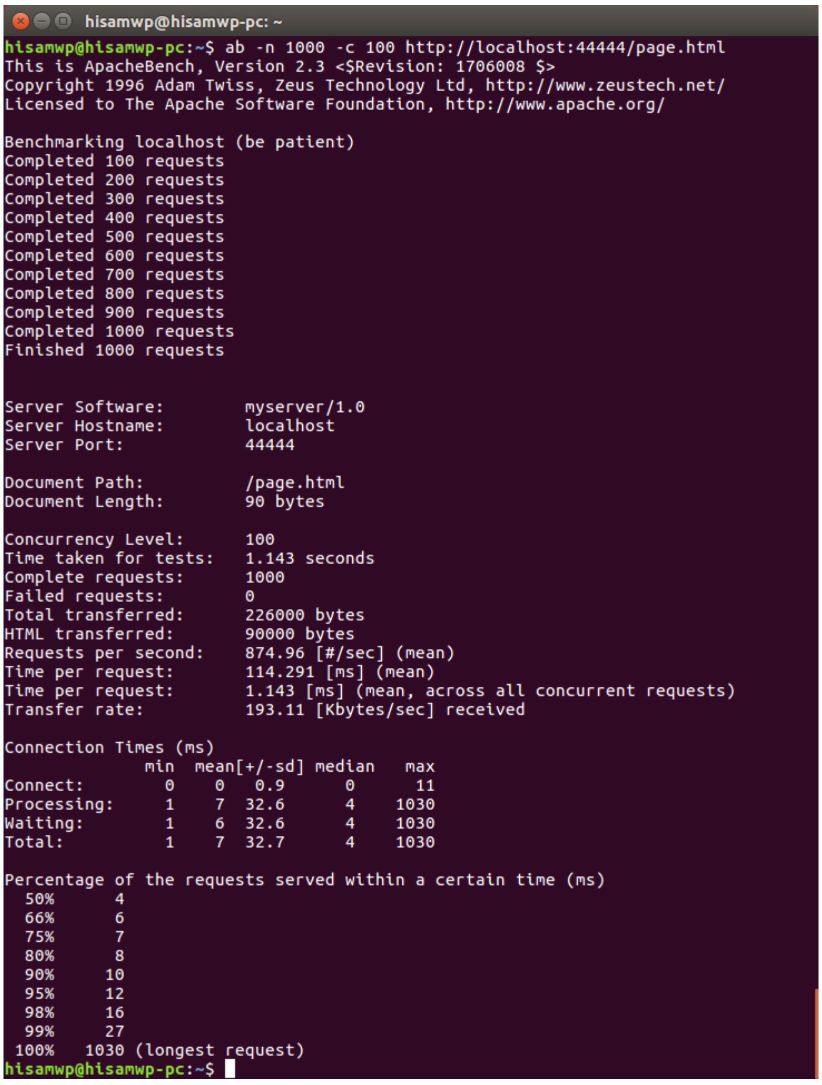
Source code lengkap dapat diakses melalui link **:**

https://github.com/WasilatulDN/PROGJAR_05111740000004/tree/master/final_project

VII. SCREENSHOT

- **Load Balancer tanpa penambahan otomatis**

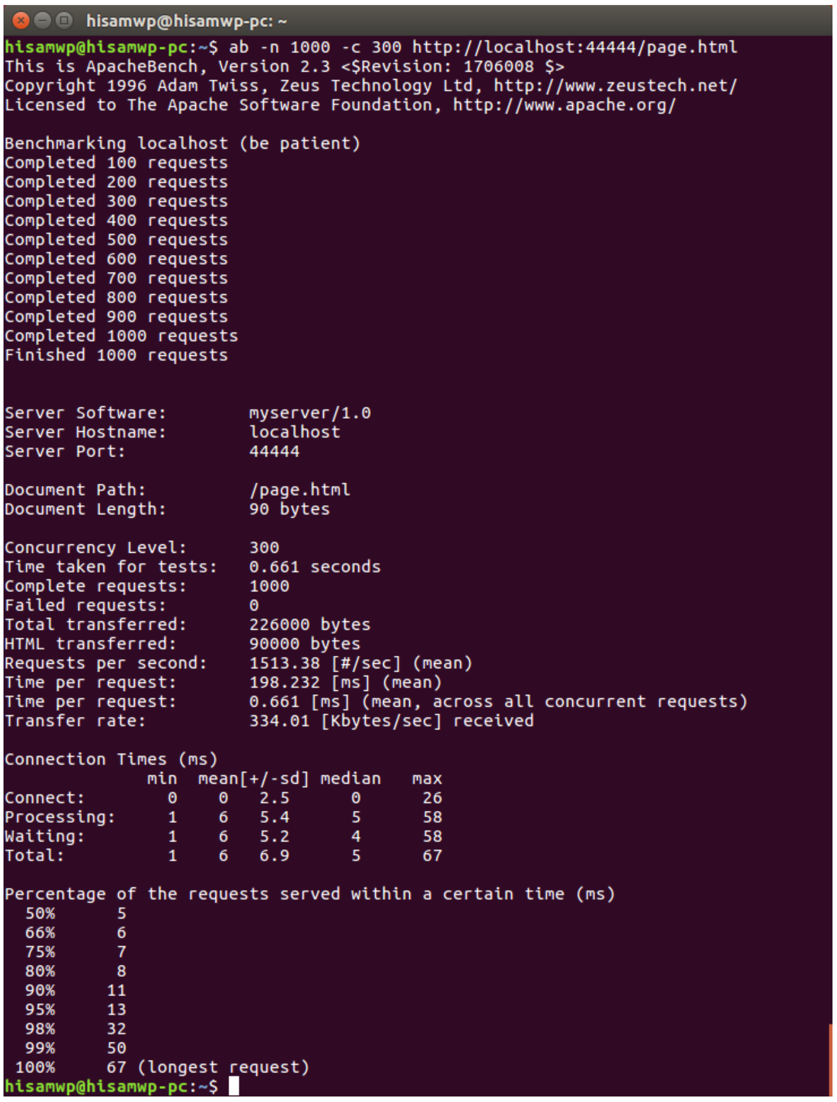
N=1000, c=100



hisamwp@hisamwp-pc: ~

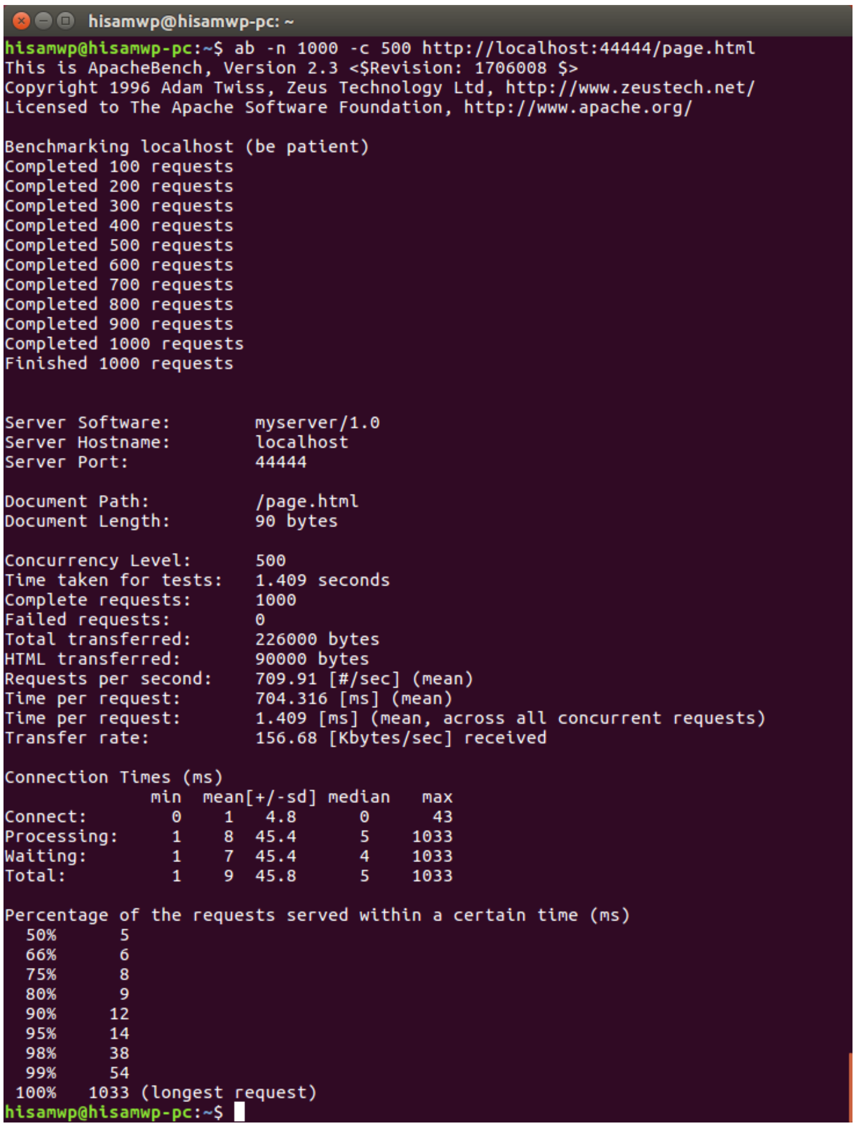
```
hisamwp@hisamwp-pc:~$ ab -n 1000 -c 100 http://localhost:44444/page.html
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        myserver/1.0
Server Hostname:        localhost
Server Port:            44444

Document Path:          /page.html
Document Length:        90 bytes

Concurrency Level:      100
Time taken for tests:   1.143 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      226000 bytes
HTML transferred:       90000 bytes
Requests per second:    874.96 [#/sec] (mean)
Time per request:       114.291 [ms] (mean)
Time per request:       1.143 [ms] (mean, across all concurrent requests)
Transfer rate:          193.11 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.9      0      11
Processing:     1    7  32.6      4    1030
Waiting:        1    6  32.6      4    1030
Total:          1    7  32.7      4    1030

Percentage of the requests served within a certain time (ms)
  50%      4
  66%      6
  75%      7
  80%      8
  90%     10
  95%     12
  98%     16
  99%     27
 100%   1030 (longest request)
hisamwp@hisamwp-pc:~$
```

N=1000, c=300



```
hisamwp@hisamwp-pc:~$ ab -n 1000 -c 300 http://localhost:44444/page.html
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        myserver/1.0
Server Hostname:        localhost
Server Port:            44444

Document Path:          /page.html
Document Length:        90 bytes

Concurrency Level:      300
Time taken for tests:   0.661 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      226000 bytes
HTML transferred:       90000 bytes
Requests per second:    1513.38 [#/sec] (mean)
Time per request:       198.232 [ms] (mean)
Time per request:       0.661 [ms] (mean, across all concurrent requests)
Transfer rate:          334.01 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   2.5      0      26
Processing:     1    6   5.4      5      58
Waiting:        1    6   5.2      4      58
Total:          1    6   6.9      5      67

Percentage of the requests served within a certain time (ms)
  50%      5
  66%      6
  75%      7
  80%      8
  90%     11
  95%     13
  98%     32
  99%     50
 100%     67 (longest request)
hisamwp@hisamwp-pc:~$
```

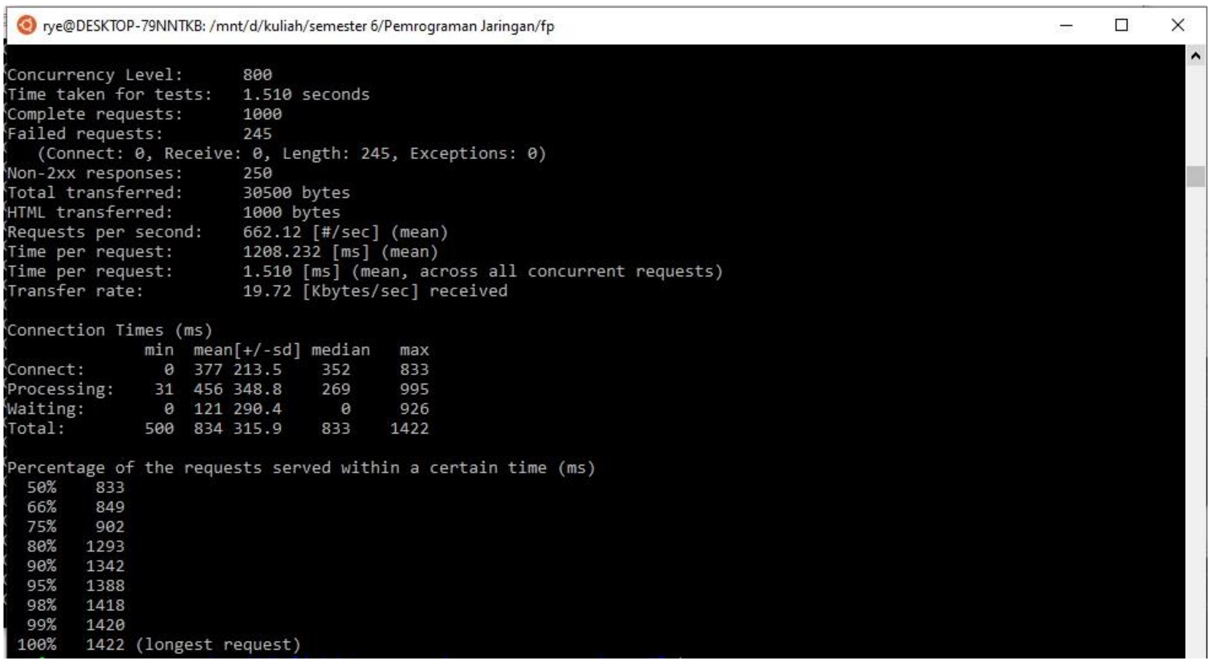
N=1000, c=500



hisamwp@hisamwp-pc: ~

```
hisamwp@hisamwp-pc:~$ ab -n 1000 -c 500 http://localhost:44444/page.html
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        myserver/1.0
Server Hostname:        localhost
Server Port:            44444

Document Path:          /page.html
Document Length:        90 bytes

Concurrency Level:      500
Time taken for tests:   1.409 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      226000 bytes
HTML transferred:       90000 bytes
Requests per second:    709.91 [#/sec] (mean)
Time per request:       704.316 [ms] (mean)
Time per request:       1.409 [ms] (mean, across all concurrent requests)
Transfer rate:          156.68 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   4.8      0      43
Processing:     1    8  45.4      5    1033
Waiting:        1    7  45.4      4    1033
Total:          1    9  45.8      5    1033

Percentage of the requests served within a certain time (ms)
  50%      5
  66%      6
  75%      8
  80%      9
  90%     12
  95%     14
  98%     38
  99%     54
 100%   1033 (longest request)
hisamwp@hisamwp-pc:~$ 
```

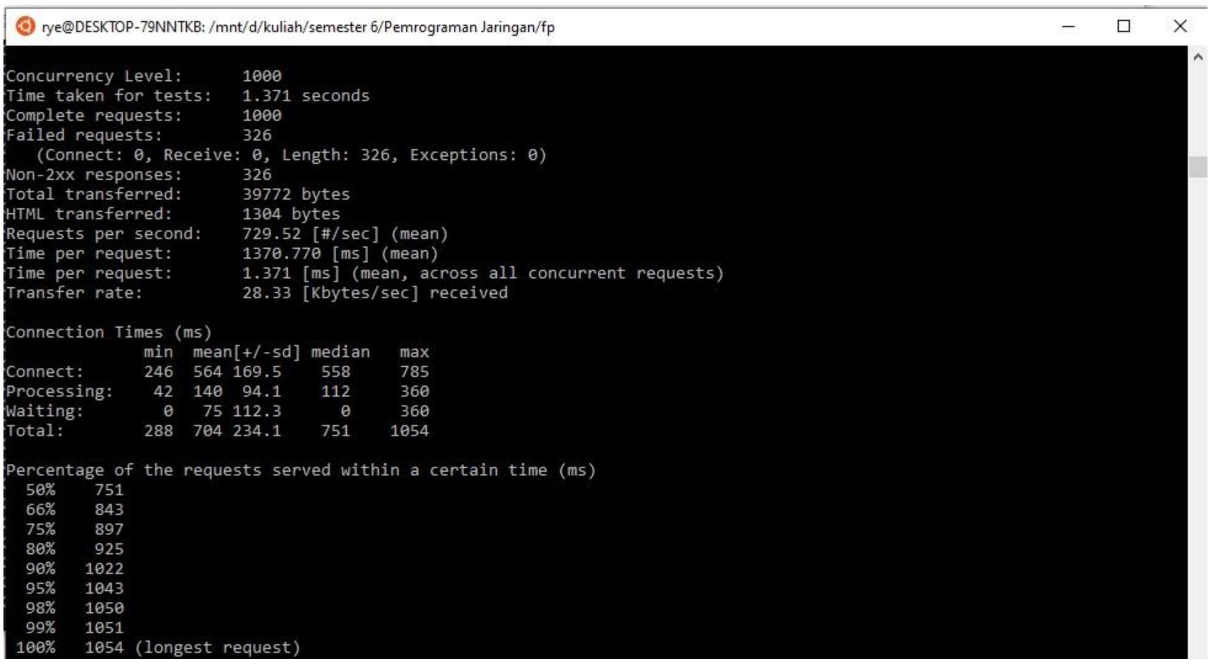
N=1000, c=800



rye@DESKTOP-79NNTKB: /mnt/d/kuliah/semester 6/Pemrograman Jaringan/fp

```
Concurrency Level:      800
Time taken for tests:   1.510 seconds
Complete requests:      1000
Failed requests:        245
   (Connect: 0, Receive: 0, Length: 245, Exceptions: 0)
Non-2xx responses:      250
Total transferred:      30500 bytes
HTML transferred:       1000 bytes
Requests per second:    662.12 [#/sec] (mean)
Time per request:       1208.232 [ms] (mean)
Time per request:       1.510 [ms] (mean, across all concurrent requests)
Transfer rate:          19.72 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  377 213.5    352     833
Processing:    31  456 348.8    269     995
Waiting:        0  121 290.4      0     926
Total:        500  834 315.9    833    1422

Percentage of the requests served within a certain time (ms)
  50%    833
  66%    849
  75%    902
  80%   1293
  90%   1342
  95%   1388
  98%   1418
  99%   1420
 100%   1422 (longest request)
```

N=1000, c=1000

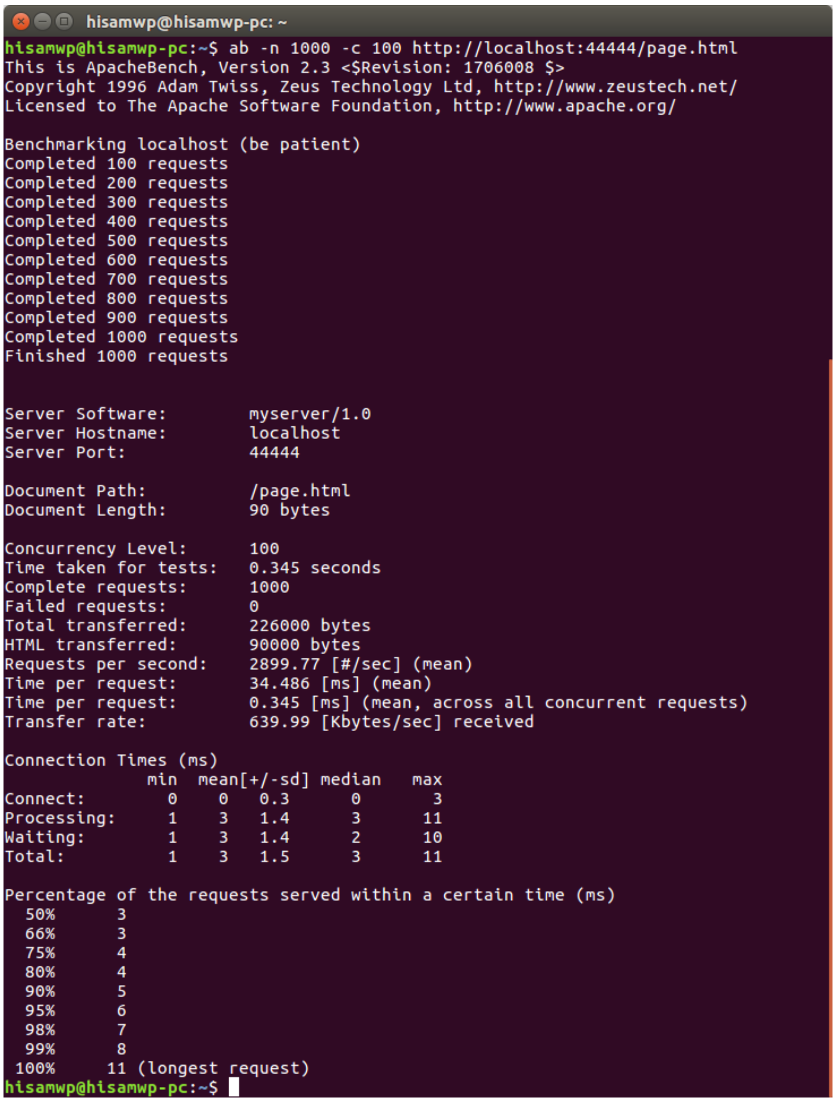


rye@DESKTOP-79NNTKB: /mnt/d/kuliah/semester 6/Pemrograman Jaringan/fp

```
Concurrency Level:      1000
Time taken for tests:   1.371 seconds
Complete requests:      1000
Failed requests:        326
   (Connect: 0, Receive: 0, Length: 326, Exceptions: 0)
Non-2xx responses:      326
Total transferred:      39772 bytes
HTML transferred:       1304 bytes
Requests per second:    729.52 [#/sec] (mean)
Time per request:       1370.770 [ms] (mean)
Time per request:       1.371 [ms] (mean, across all concurrent requests)
Transfer rate:          28.33 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      246  564 169.5    558     785
Processing:    42  140  94.1    112     360
Waiting:        0   75 112.3      0     360
Total:        288  704 234.1    751    1054

Percentage of the requests served within a certain time (ms)
  50%    751
  66%    843
  75%    897
  80%    925
  90%   1022
  95%   1043
  98%   1050
  99%   1051
 100%   1054 (longest request)
```

- **Load Balancer dengan penambahan otomatis**

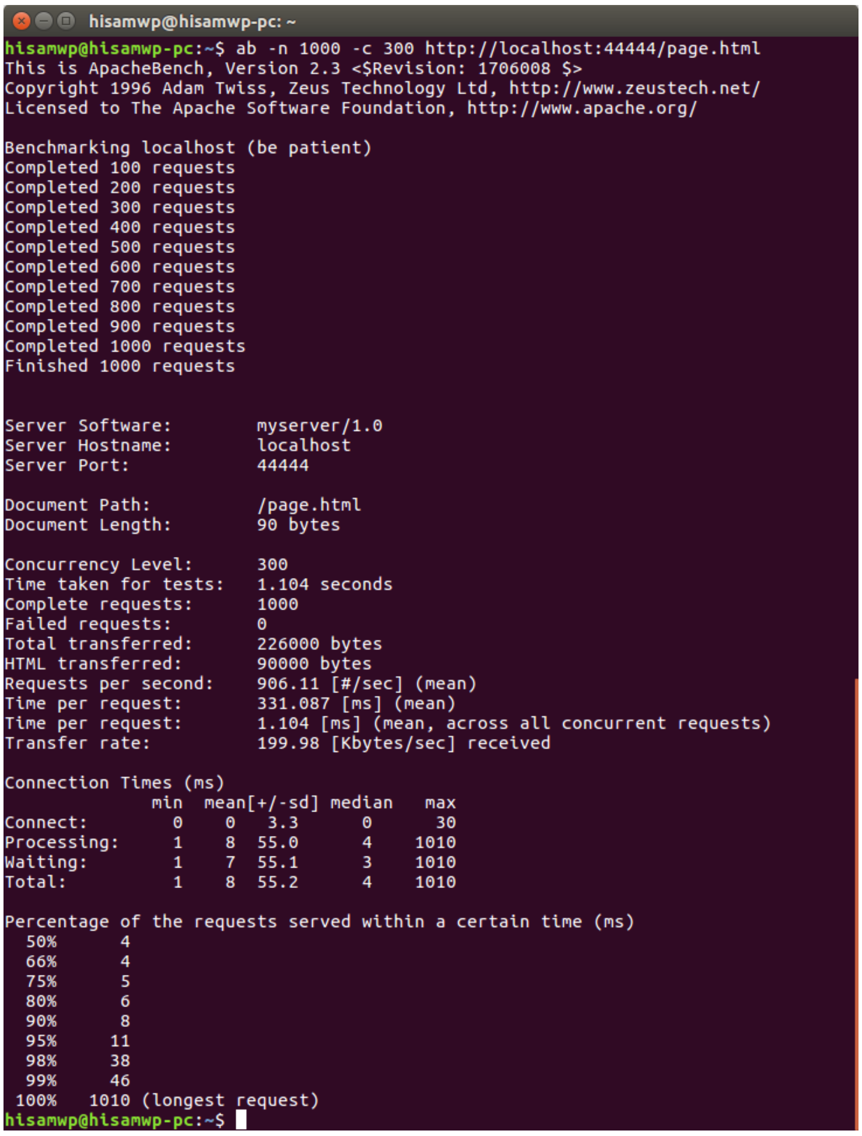
N=1000, c=100



hisamwp@hisamwp-pc: ~

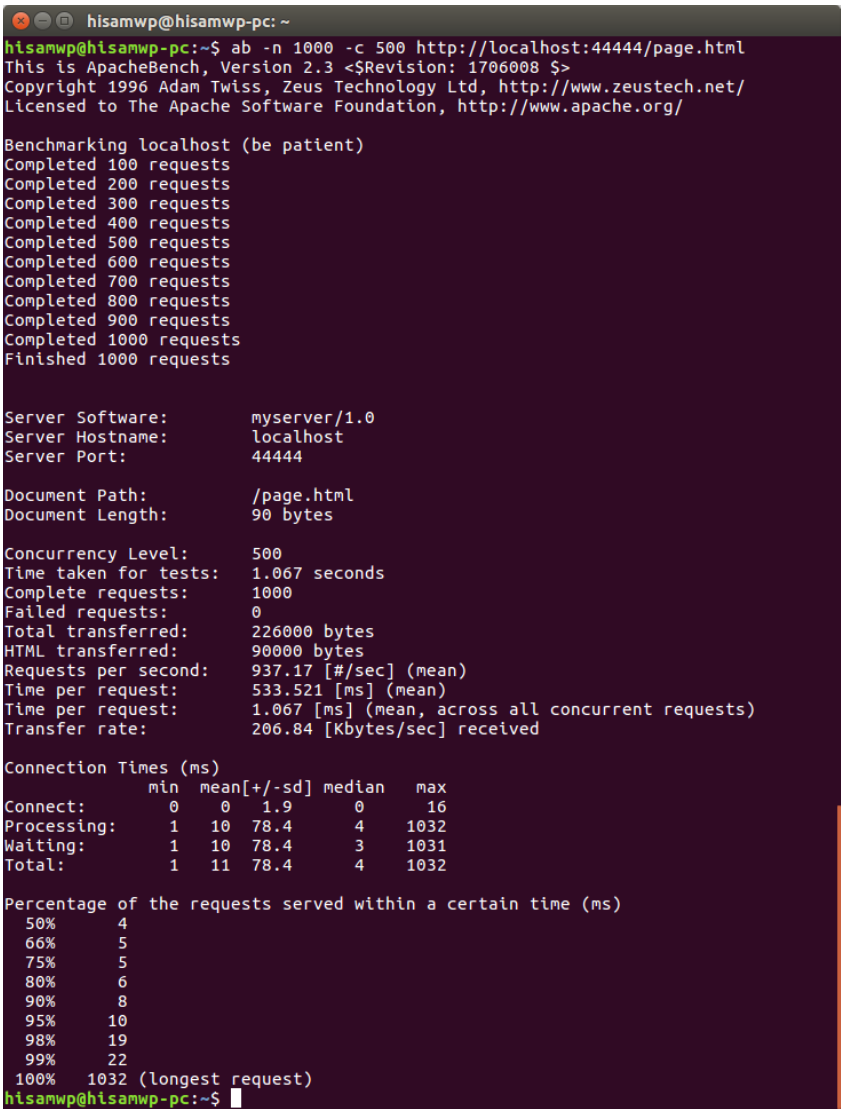
```
hisamwp@hisamwp-pc:~$ ab -n 1000 -c 100 http://localhost:44444/page.html
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        myserver/1.0
Server Hostname:        localhost
Server Port:            44444

Document Path:          /page.html
Document Length:        90 bytes

Concurrency Level:      100
Time taken for tests:   0.345 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      226000 bytes
HTML transferred:       90000 bytes
Requests per second:    2899.77 [#/sec] (mean)
Time per request:       34.486 [ms] (mean)
Time per request:       0.345 [ms] (mean, across all concurrent requests)
Transfer rate:          639.99 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       3
Processing:     1    3   1.4      3      11
Waiting:        1    3   1.4      2      10
Total:          1    3   1.5      3      11

Percentage of the requests served within a certain time (ms)
  50%      3
  66%      3
  75%      4
  80%      4
  90%      5
  95%      6
  98%      7
  99%      8
 100%     11 (longest request)
hisamwp@hisamwp-pc:~$ 
```

N=1000, c=300



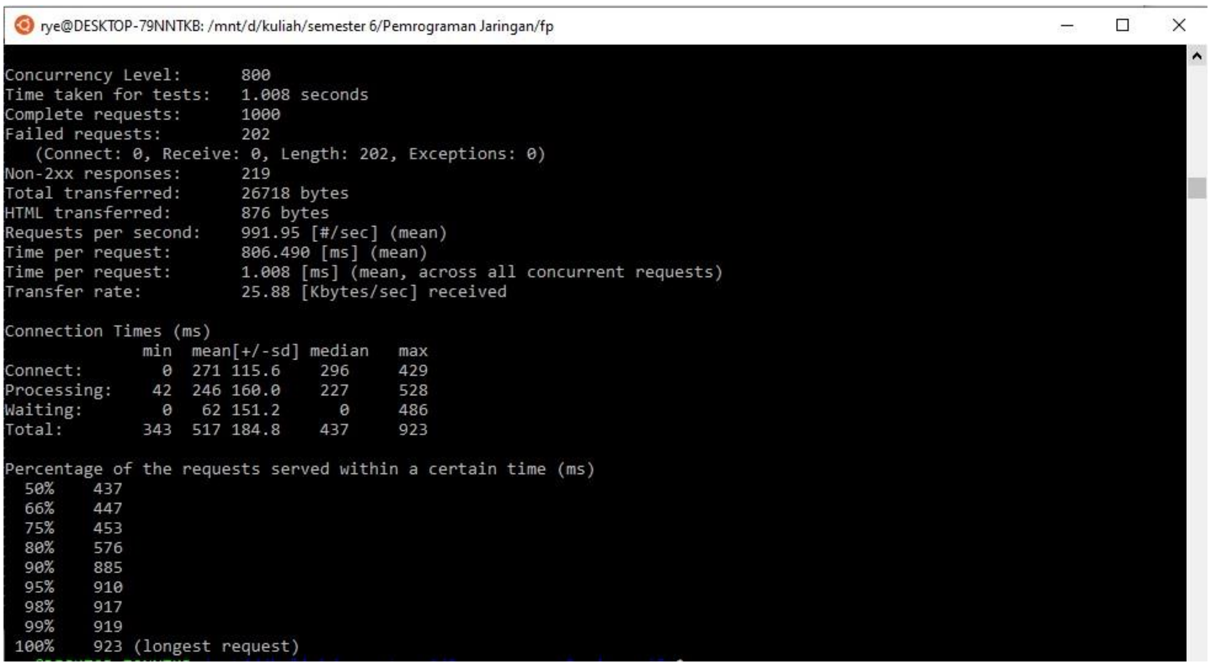
```
hisamwp@hisamwp-pc:~$ ab -n 1000 -c 300 http://localhost:44444/page.html
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        myserver/1.0
Server Hostname:        localhost
Server Port:            44444

Document Path:          /page.html
Document Length:        90 bytes

Concurrency Level:      300
Time taken for tests:   1.104 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      226000 bytes
HTML transferred:       90000 bytes
Requests per second:    906.11 [#/sec] (mean)
Time per request:       331.087 [ms] (mean)
Time per request:       1.104 [ms] (mean, across all concurrent requests)
Transfer rate:          199.98 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   3.3      0      30
Processing:     1    8  55.0      4    1010
Waiting:        1    7  55.1      3    1010
Total:          1    8  55.2      4    1010

Percentage of the requests served within a certain time (ms)
  50%      4
  66%      4
  75%      5
  80%      6
  90%      8
  95%     11
  98%     38
  99%     46
 100%   1010 (longest request)
hisamwp@hisamwp-pc:~$
```

N=1000, c=500



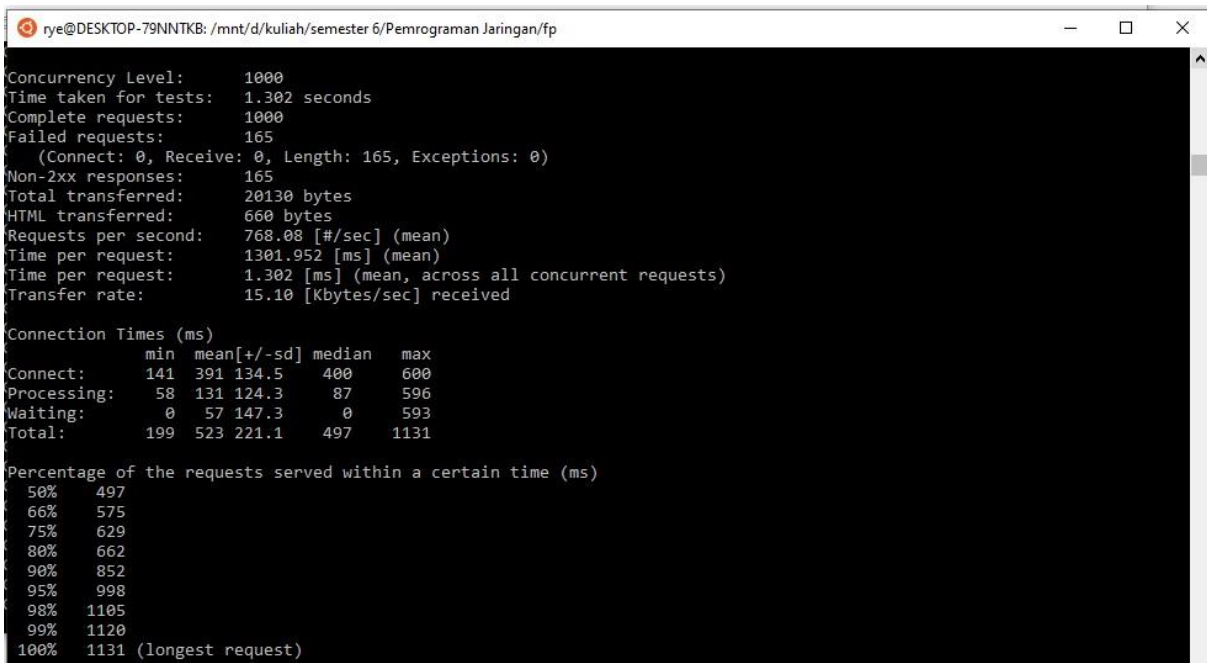
```
hisamwp@hisamwp-pc:~$ ab -n 1000 -c 500 http://localhost:44444/page.html
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        myserver/1.0
Server Hostname:        localhost
Server Port:            44444

Document Path:          /page.html
Document Length:        90 bytes

Concurrency Level:      500
Time taken for tests:   1.067 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      226000 bytes
HTML transferred:       90000 bytes
Requests per second:    937.17 [#/sec] (mean)
Time per request:       533.521 [ms] (mean)
Time per request:       1.067 [ms] (mean, across all concurrent requests)
Transfer rate:          206.84 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.9      0      16
Processing:     1   10  78.4      4    1032
Waiting:        1   10  78.4      3    1031
Total:          1   11  78.4      4    1032

Percentage of the requests served within a certain time (ms)
  50%      4
  66%      5
  75%      5
  80%      6
  90%      8
  95%     10
  98%     19
  99%     22
 100%   1032 (longest request)
hisamwp@hisamwp-pc:~$
```

N=1000, c=800


rye@DESKTOP-79NNTKB: /mnt/d/kuliah/semester 6/Pemrograman Jaringan/fp

```
Concurrency Level:      800
Time taken for tests:   1.008 seconds
Complete requests:      1000
Failed requests:        202
   (Connect: 0, Receive: 0, Length: 202, Exceptions: 0)
Non-2xx responses:      219
Total transferred:      26718 bytes
HTML transferred:       876 bytes
Requests per second:    991.95 [#/sec] (mean)
Time per request:       806.490 [ms] (mean)
Time per request:       1.008 [ms] (mean, across all concurrent requests)
Transfer rate:          25.88 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  271 115.6    296     429
Processing:    42  246 160.0    227     528
Waiting:        0   62 151.2      0     486
Total:        343  517 184.8    437     923

Percentage of the requests served within a certain time (ms)
  50%    437
  66%    447
  75%    453
  80%    576
  90%    885
  95%    910
  98%    917
  99%    919
 100%    923 (longest request)
```

N=1000, c=1000


rye@DESKTOP-79NNTKB: /mnt/d/kuliah/semester 6/Pemrograman Jaringan/fp

```
Concurrency Level:      1000
Time taken for tests:   1.302 seconds
Complete requests:      1000
Failed requests:        165
   (Connect: 0, Receive: 0, Length: 165, Exceptions: 0)
Non-2xx responses:      165
Total transferred:      20130 bytes
HTML transferred:       660 bytes
Requests per second:    768.08 [#/sec] (mean)
Time per request:       1301.952 [ms] (mean)
Time per request:       1.302 [ms] (mean, across all concurrent requests)
Transfer rate:          15.10 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      141  391 134.5    400     600
Processing:    58  131 124.3     87     596
Waiting:        0   57 147.3      0     593
Total:        199  523 221.1    497    1131

Percentage of the requests served within a certain time (ms)
  50%    497
  66%    575
  75%    629
  80%    662
  90%    852
  95%    998
  98%   1105
  99%   1120
 100%   1131 (longest request)
```