Faculty of Engineering, Alexandria University

Computer and Systems Engineering Department

Computer Communications and Networks: Fall 2018

Assignment 1

Amira nabil ( )

Amr Mohamed NasrEldin (47)

Michael Raafat Mikhail (57)

Overall Organization:

We have divided the project into six main folders

- parser:
  - has parser for the file given to client for requests.
- Client:
  - has the client logic (create socket, establish connection, read and send requests, get responses).
- Server:
  - has the server logic (start server, update timeout, clean workers).
- File system:
  - has reading from and writing to files logic.
- Web models:
  - has network enums, request and response classes along with a class wrapping a request and it's port number.
- Utils:
  - has util classes for string manipulations, and socket operations.
- Main:
  - has the main classes of client and server.

Functions:

Client functions:

- Start_client:
  - gets requests read from file and calls send_message.
- Create_socket:
  - creates client socket.
- Get_socket:
  - creates new socket or fetch already created socket from sockets map.
- Connect_server:
  - connects to server if possible.
- Send_message:
  - sends request to server if possible.
- Receive_response: receive response from server if possible.

Server functions:

- Start_server:
  - create listen socket and binds it to the server address, and starts waiting for clients to serve.

- Clean_workers :
    - checks if a client has finished or maximum workers limit exceeded, in both cases a thread is killed.

- Update_timeouts :
    - time out is function of workers size. when a worker is added or cleaned, all timeouts of all workers are updated.

Data structures:

Web Models:

- Request :
    - holds request and all its details.
- Response :
    - holds response and all its details.
- Network Enum :
    - holds type and protocol of requests and responses.

Client used data structures:

- map <string, int> myConnections :
    - saves each socket for specific ip address corresponding to host name and port number.
- vector <pair<int, struct sockaddr_in>> socket_map:
    - saves struct of sockaddr_in and socket for each new request.
- requestAndPortNo object:
    - wrapper object that wraps each request with the port number.

Server used data structures:

- clientWorker object :
    - it handles a client, create a thread to serve it , set it's timeout, set if finished, start serving and kills the thread.
- Vector of client workers:
    - Keep instance of all current client workers.

Important Assumptions & Notes about the implementation:

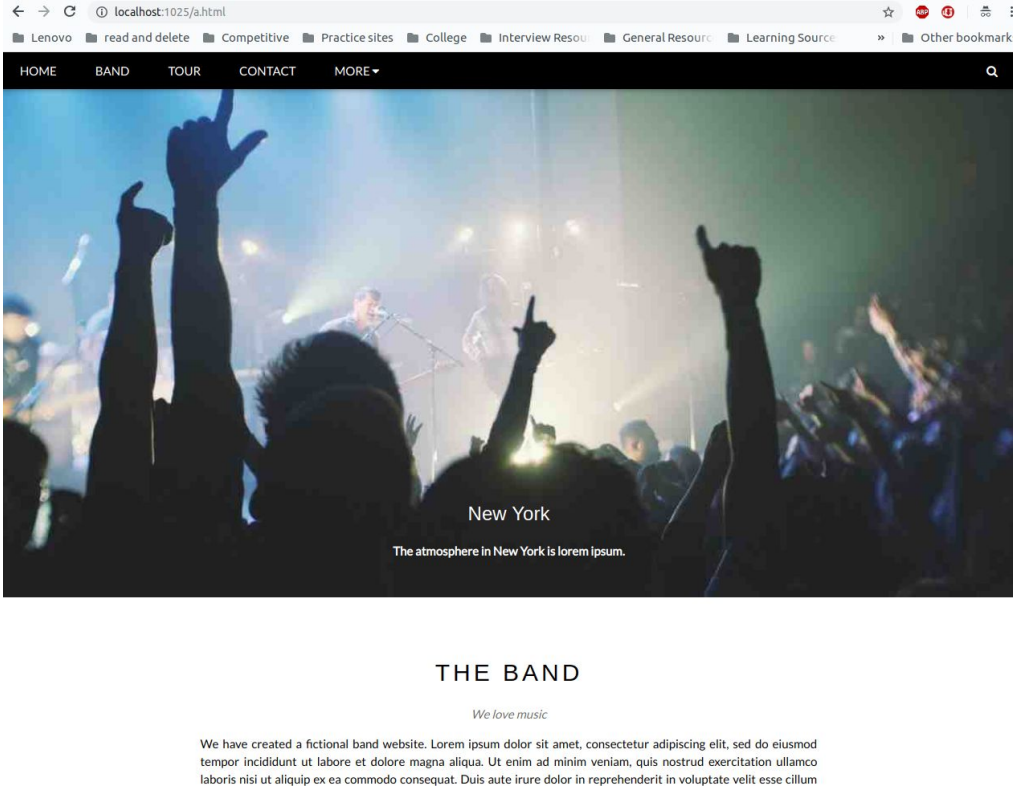
- ## For Client:

  1- using pipelining for GET requests only, on POST we send request and start receive all previous requests.
  2- we have a limit for pipeline to have **20** requests to send then we receive all requests.
  3- for each command we get host name and optional port number, we set 80 for port number as default if there is no port number provided in command.

- ## For Server:

  1- use threads instead of processes to handle client requests, since threads are faster to create, and share memory with the original process.
  2- ask threads to terminate when reaching full capacity.
  3- server won't serve post requests with no Content-Length header.
  4- timeout starts with 2 minute interval, reduces with each client.
  5- no busy wait, blocked wait until timeout or shutdown.

Using Real Browser To Test Server



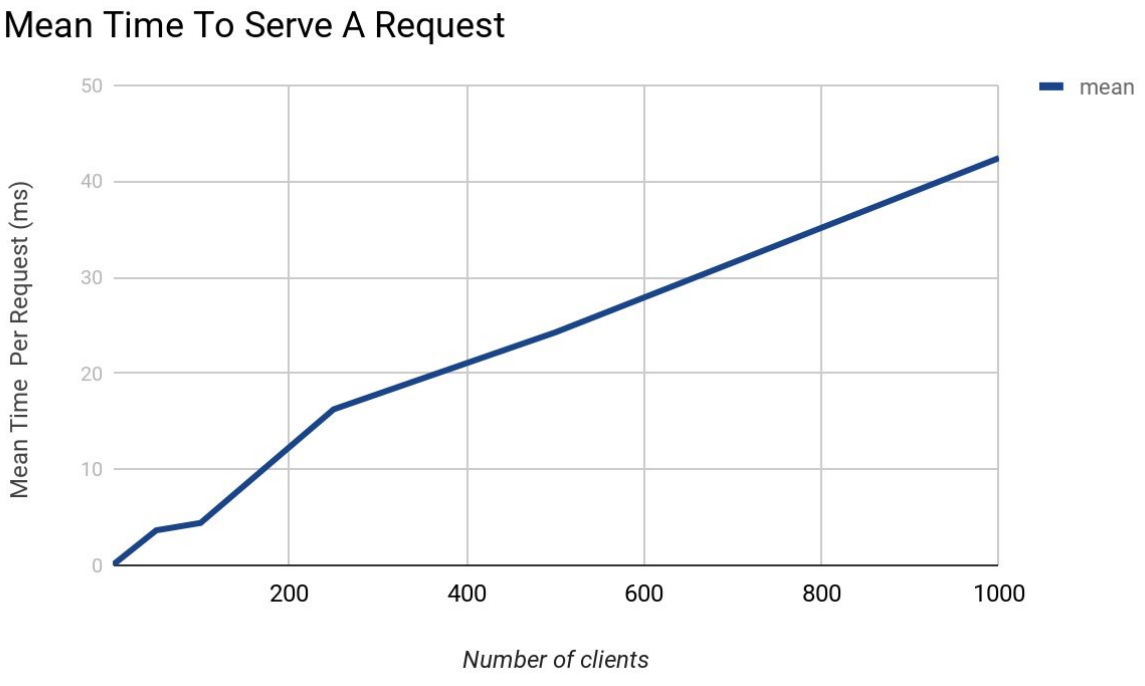
THE BAND

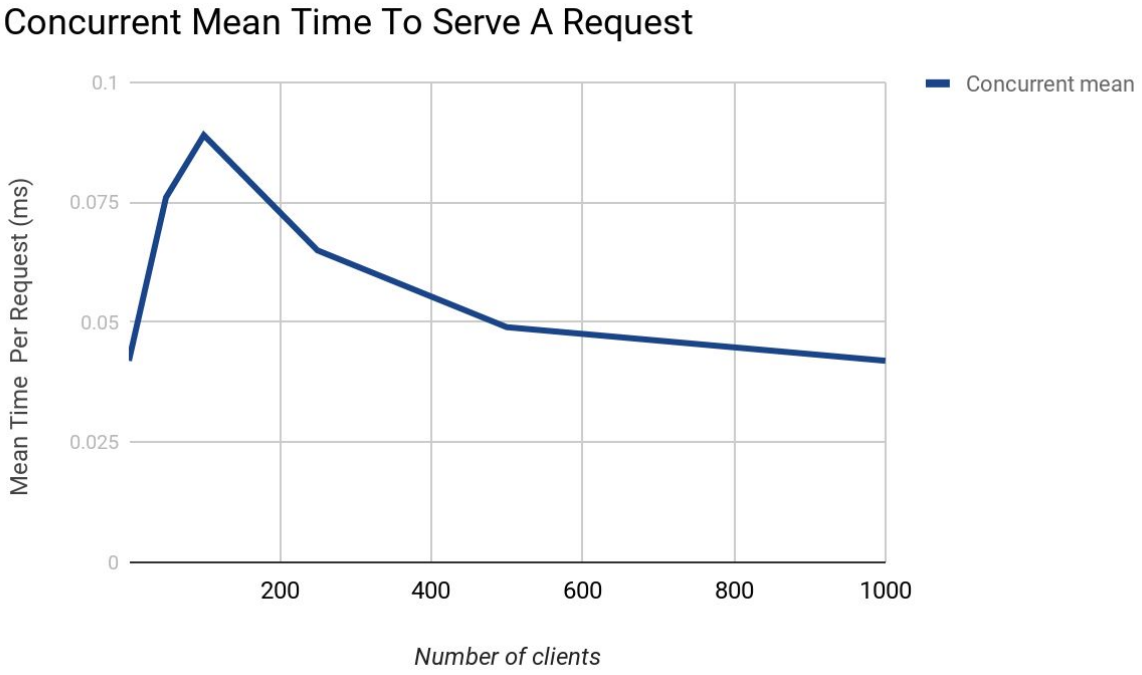
We love music

We have created a fictional band website. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum

```
GET /w3images/bandmember.jpg HTTP/1.1
Accept : image/webp,image/apng,image/*,*/*;q=0.8
Accept-Encoding : gzip, deflate, br
Accept-Language : en-GB,en-US;q=0.9,en;q=0.8
Connection : keep-alive
Host : localhost:1025
Referer : http://localhost:1025/a.html
User-Agent : Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36

GET /w3images/newyork.jpg HTTP/1.1
Accept : image/webp,image/apng,image/*,*/*;q=0.8
Accept-Encoding : gzip, deflate, br
Accept-Language : en-GB,en-US;q=0.9,en;q=0.8
Connection : keep-alive
Host : localhost:1025
Referer : http://localhost:1025/a.html
User-Agent : Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36

GET /w3images/paris.jpg HTTP/1.1
Accept : image/webp,image/apng,image/*,*/*;q=0.8
Accept-Encoding : gzip, deflate, br
Accept-Language : en-GB,en-US;q=0.9,en;q=0.8
Connection : keep-alive
Host : localhost:1025
Referer : http://localhost:1025/a.html
User-Agent : Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36

GET /w3images/sanfran.jpg HTTP/1.1
Accept : image/webp,image/apng,image/*,*/*;q=0.8
Accept-Encoding : gzip, deflate, br
Accept-Language : en-GB,en-US;q=0.9,en;q=0.8
Connection : keep-alive
Host : localhost:1025
Referer : http://localhost:1025/a.html
User-Agent : Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36

GET /w3images/map.jpg HTTP/1.1
Accept : image/webp,image/apng,image/*,*/*;q=0.8
Accept-Encoding : gzip, deflate, br
Accept-Language : en-GB,en-US;q=0.9,en;q=0.8
Connection : keep-alive
Host : localhost:1025
Referer : http://localhost:1025/a.html
User-Agent : Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36

GET /favicon.ico HTTP/1.1
Accept : image/webp,image/apng,image/*,*/*;q=0.8
Accept-Encoding : gzip, deflate, br
Accept-Language : en-GB,en-US;q=0.9,en;q=0.8
Connection : keep-alive
Host : localhost:1025
Referer : http://localhost:1025/a.html
User-Agent : Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
```

Performance Evaluation




Concurrent Mean Time To Serve A Request
Concurrent mean
0.1
0.075
0.05
0.025
0
Mean Time Per Request (ms)
200
400
600
800
1000
Number of clients