

Opis doświadczenia

Celem doświadczenia jest:

1. Uruchomić i przeanalizować działanie skryptu serwera http.
2. Nawiązać połączenie z przeglądarką internetową.
3. Zaprezentować skrypt tak aby wysyłał do klienta nagłówek jego żądania.
4. Zaprezentować skrypt, aby obsługiwał żądania klienta do prostego tekstowego serwisu WWW (kilka statycznych stron z wzajemnymi odwołaniami) zapisanego w pewnym katalogu dysku lokalnego komputera, na którym uruchomiony jest skrypt serwera.
5. Przechwyć komunikaty do/od serwera za pomocą analizatora sieciowego - przeanalizować ich konstrukcję.

Realizacja zadania

Aby uruchomić skrypt musimy najpierw dostosować adres hosta do naszych potrzeb. Tutaj zmienimy go na **localhost**/127.0.0.1. Przy okazji zmienię port na 8888, jednak ten ruch nie jest konieczny, aby uruchomić serwer. Program uruchamiamy wpisując **perl server3.pl** w terminalu. Skrypt to prosta implementacja serwera http. W pierwszych liniach deklarujemy adres oraz port, na którym nasłuchujemy. Następnie przechodzimy do zewnętrznej pętli, która działa, jeżeli połączenie z klientem zostało nawiązane. W wewnętrznej pętli nasłuchujemy zapytań od klienta (metodą **get_request**). Następnie sprawdzamy jaki to był rodzaj zapytania, jeżeli było to zapytanie "GET" wówczas zapisujemy w zmiennej naszą stronę index.html i wysyłamy w postaci odpowiedzi od serwera metodą **send_file_response**. Nasz **index.html** jest już widoczny pod wskazanym adresem i portem. Jeżeli zapytanie od klienta jest inne niż "**GET**" wówczas serwer wysyła błąd,

który pokazuje się jako wiadomość tekstowa pod wskazanym adresem.

W celu nawiązanie kontaktu z przeglądarką do folderu, w którym znajduje się serwer dołączamy prosty plik index.html z jakimś podstawowym szablonem znaczników. Po uruchomieniu serwera oraz wpisaniu w oknie przeglądarki **localhost:8888** w przeglądarce ukazuje nam się nasza prosta strona.

Teraz zaprezentujemy skrypt, który odeśle nam na stronę nagłówki zapytania. W tym celu napiszmy prosty kod pythonie implementujący serwer http, jednak nadpiszmy metodę **do_GET**, w której użyjemy metod **wfile.write**, które wypiszą nam wartość tekstową argumentu jaki przyjmą. Jako argument podajemy wbudowaną metodę **headers**.

```
from http.server import HTTPServer, BaseHTTPRequestHandler

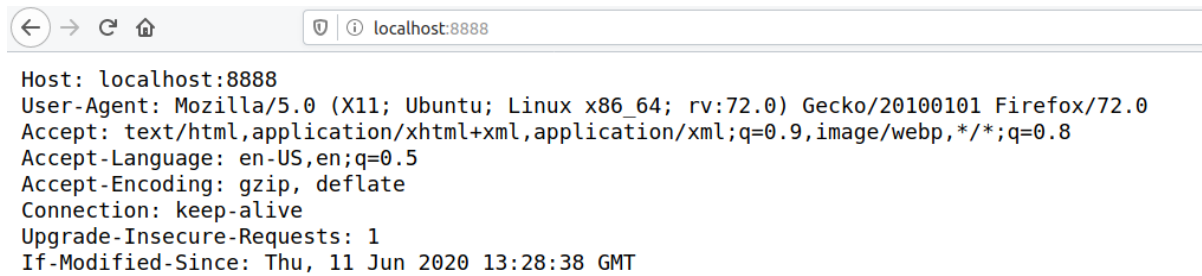
class RequestHandler(BaseHTTPRequestHandler):
    """zad 3"""

    def do_GET(self):
        if self.path.endswith('/'):
            self.send_response(200)
            """print headers in html on port 8888 """
            self.wfile.write(str(self.headers).encode())

def main():
    PORT = 8888
    server_address = ('localhost', PORT)
    server = HTTPServer(server_address, RequestHandler)
    print("server running. Port ", PORT, "\n")
    server.serve_forever()

if __name__ == '__main__':
    main()
```

Wszystko realizujemy na tym samym porcie co w poprzednim przykładzie (localhost:8888).



A screenshot of a web browser window. The address bar shows 'localhost:8888'. Below the address bar, the following HTTP headers are displayed:

```
Host: localhost:8888
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:72.0) Gecko/20100101 Firefox/72.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-Modified-Since: Thu, 11 Jun 2020 13:28:38 GMT
```

Teraz możemy odczytać wszystkie nagłówki. Widzimy, że jest ich 8 i są wylistowane każdy od nowej linii.

Teraz zaprezentujemy skrypt implementujący serwer http i sprawmy, aby obsługiwał on połączenie z naszą stroną. W kodzie nie zamieszczamy lokalizacji naszego pliku index.html, bo gdy taki plik w folderze z serwerem jest jeden on automatycznie go rozpozna.



```
import http.server
import socketserver

def main():
    """zad 4"""

    PORT = 8888
    Handler = http.server.SimpleHTTPRequestHandler
    httpd = socketserver.TCPServer("", PORT), Handler)
    print("serving at port", PORT)
    httpd.serve_forever()

if __name__ == '__main__':
    main()
```

Teraz zaprezentujemy prostą stronę, która będzie zawierała 3 inne podstrony wyświetlające zdjęcie oraz dające możliwość powrotu do strony głównej.

```

<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>VOLLEYBALL CLUBS</title>
  <style>
    body{
      display:block;
      background-color:yellow;
      float:left;
      font-family:verdana;
      font-size:300%;
    }
  </style>
</head>
<body>
  <h1>FIND LOGO</h1>
  
  <a href="zenit.html">ZENIT</a>
  <a href="skra.html">SKRA </a>
  <a href="zaksa.html">ZAKSA</a>
</body>
</html>

```

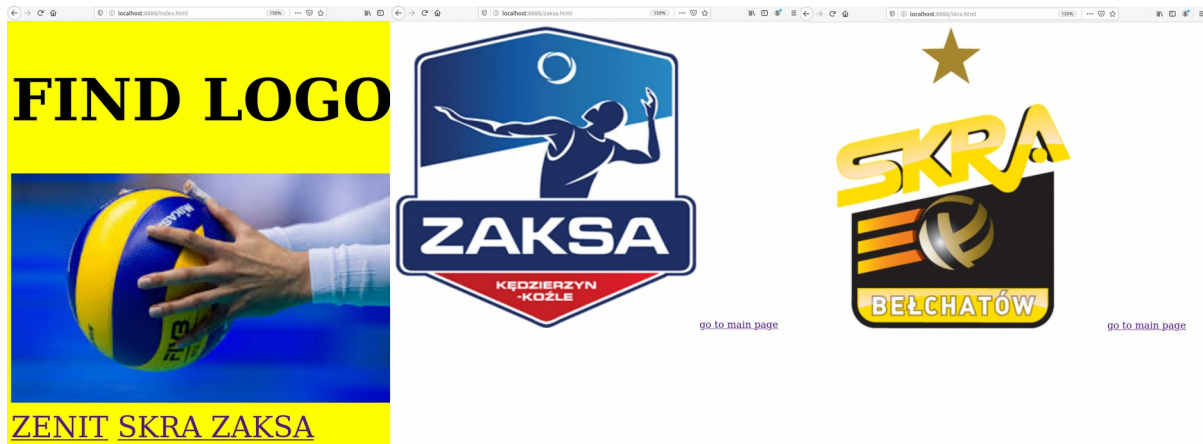
Podstrony ze zdjęciami są identyczne i zachowane w formacie:

```

<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>ZAKSA KEDZIERZYN-KOZLE</title>
</head>
<body>
  
  <a href="index.html">go to main page </a></p>
</body>
</html>

```

Po uruchomieniu serwera i odpaleniu w przeglądarce localhost-a na zadanym porcie nasza strona główna i dwie przykładowe podstrony prezentują się następująco.

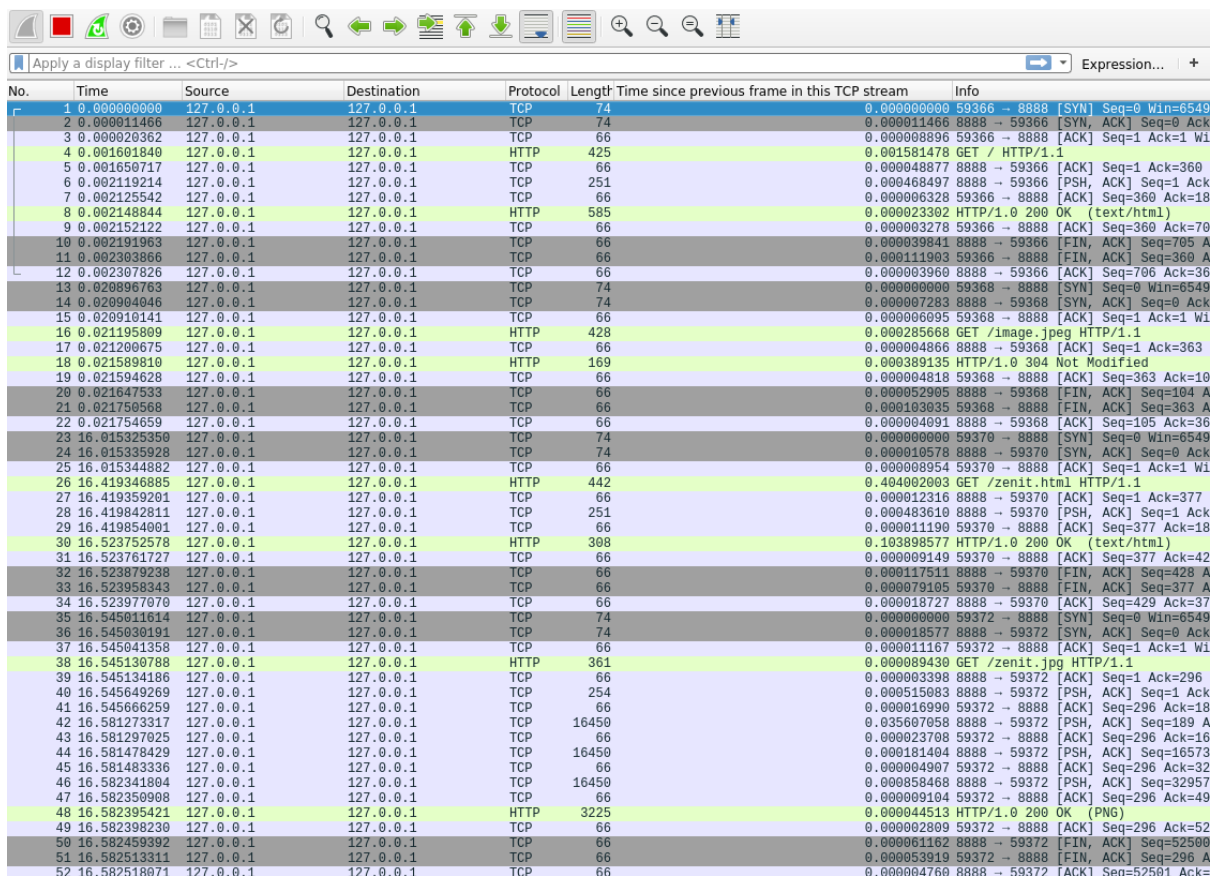


Teraz odpalmy wireshark-a z opcją loopback, żeby przechwycić komunikaty od serwera.

Po odświeżeniu strony w wireshark-u otrzymujemy:

No.	Time	Source	Destination	Protocol	Length	Time since previous frame in this TCP stream	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	0.000000000	59366 → 8888 [SYN] Seq=0 Win=6549
2	0.000011466	127.0.0.1	127.0.0.1	TCP	74	0.000011466	8888 → 59366 [SYN, ACK] Seq=0 Ack
3	0.000020362	127.0.0.1	127.0.0.1	TCP	66	0.000000896	59366 → 8888 [ACK] Seq=1 Ack=1 Wi
4	0.001601840	127.0.0.1	127.0.0.1	HTTP	425	0.001581478	GET / HTTP/1.1
5	0.001650717	127.0.0.1	127.0.0.1	TCP	66	0.000048877	8888 → 59366 [ACK] Seq=1 Ack=360
6	0.002119214	127.0.0.1	127.0.0.1	TCP	251	0.000468497	8888 → 59366 [PSH, ACK] Seq=1 Ack
7	0.002125542	127.0.0.1	127.0.0.1	TCP	66	0.000006328	59366 → 8888 [ACK] Seq=360 Ack=18
8	0.002148844	127.0.0.1	127.0.0.1	HTTP	585	0.000023302	HTTP/1.0 200 OK (text/html)
9	0.002152122	127.0.0.1	127.0.0.1	TCP	66	0.000003278	59366 → 8888 [ACK] Seq=360 Ack=70
10	0.002191963	127.0.0.1	127.0.0.1	TCP	66	0.000039841	8888 → 59366 [FIN, ACK] Seq=705 A
11	0.002308366	127.0.0.1	127.0.0.1	TCP	66	0.000111903	59366 → 8888 [FIN, ACK] Seq=360 A
12	0.002307826	127.0.0.1	127.0.0.1	TCP	66	0.000003960	8888 → 59366 [ACK] Seq=706 Ack=36
13	0.000000000	127.0.0.1	127.0.0.1	TCP	74	0.000000000	59366 → 8888 [SYN] Seq=0 Win=6549
14	0.000000000	127.0.0.1	127.0.0.1	TCP	74	0.000000000	8888 → 59366 [SYN, ACK] Seq=0 Ack
15	0.000000000	127.0.0.1	127.0.0.1	TCP	66	0.000000000	59366 → 8888 [ACK] Seq=1 Ack=1 Wi
16	0.0021195809	127.0.0.1	127.0.0.1	HTTP	428	0.000285668	GET /image.jpeg HTTP/1.1
17	0.0021200675	127.0.0.1	127.0.0.1	TCP	66	0.000004866	8888 → 59366 [ACK] Seq=1 Ack=363
18	0.0021589810	127.0.0.1	127.0.0.1	HTTP	169	0.000389135	HTTP/1.0 304 Not Modified
19	0.0021594628	127.0.0.1	127.0.0.1	TCP	66	0.000004818	59366 → 8888 [ACK] Seq=363 Ack=10
20	0.0021647533	127.0.0.1	127.0.0.1	TCP	66	0.000052905	8888 → 59366 [FIN, ACK] Seq=104 A
21	0.0021750568	127.0.0.1	127.0.0.1	TCP	66	0.000103035	59366 → 8888 [FIN, ACK] Seq=363 A
22	0.0021754659	127.0.0.1	127.0.0.1	TCP	66	0.000004091	8888 → 59366 [ACK] Seq=105 Ack=36

Teraz przejdźmy do jednej z podstron.



No.	Time	Source	Destination	Protocol	Length	Time since previous frame in this TCP stream	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	0.000000000	59366 → 8888 [SYN] Seq=0 Win=6549
2	0.000011466	127.0.0.1	127.0.0.1	TCP	74	0.000011466	8888 → 59366 [SYN, ACK] Seq=0 Ack=
3	0.000020362	127.0.0.1	127.0.0.1	TCP	66	0.000008896	59366 → 8888 [ACK] Seq=1 Ack=1 Wi
4	0.001601840	127.0.0.1	127.0.0.1	HTTP	425	0.001581478	GET / HTTP/1.1
5	0.001650717	127.0.0.1	127.0.0.1	TCP	66	0.000048877	8888 → 59366 [ACK] Seq=1 Ack=360
6	0.002119214	127.0.0.1	127.0.0.1	TCP	251	0.000468497	8888 → 59366 [PSH, ACK] Seq=1 Ack=
7	0.002125542	127.0.0.1	127.0.0.1	TCP	66	0.000066328	59366 → 8888 [ACK] Seq=360 Ack=18
8	0.002148844	127.0.0.1	127.0.0.1	HTTP	585	0.000023302	HTTP/1.0 200 OK (text/html)
9	0.002152122	127.0.0.1	127.0.0.1	TCP	66	0.000003278	59366 → 8888 [ACK] Seq=360 Ack=70
10	0.002191963	127.0.0.1	127.0.0.1	TCP	66	0.000039641	8888 → 59366 [FIN, ACK] Seq=705 A
11	0.002303866	127.0.0.1	127.0.0.1	TCP	66	0.000111903	59366 → 8888 [FIN, ACK] Seq=360 A
12	0.002307826	127.0.0.1	127.0.0.1	TCP	66	0.000003960	8888 → 59366 [ACK] Seq=706 Ack=36
13	0.020896763	127.0.0.1	127.0.0.1	TCP	74	0.000000000	59368 → 8888 [SYN] Seq=0 Win=6549
14	0.020904046	127.0.0.1	127.0.0.1	TCP	74	0.000007283	8888 → 59368 [SYN, ACK] Seq=0 Ack=
15	0.020910141	127.0.0.1	127.0.0.1	TCP	66	0.000006095	59368 → 8888 [ACK] Seq=1 Ack=1 Wi
16	0.021195809	127.0.0.1	127.0.0.1	HTTP	428	0.000285668	GET /image.jpeg HTTP/1.1
17	0.021200675	127.0.0.1	127.0.0.1	TCP	66	0.000004866	8888 → 59368 [ACK] Seq=1 Ack=363
18	0.021589810	127.0.0.1	127.0.0.1	HTTP	169	0.000389135	HTTP/1.0 304 Not Modified
19	0.021594628	127.0.0.1	127.0.0.1	TCP	66	0.000004818	59368 → 8888 [ACK] Seq=363 Ack=10
20	0.021647533	127.0.0.1	127.0.0.1	TCP	66	0.000052905	8888 → 59368 [FIN, ACK] Seq=104 A
21	0.021750568	127.0.0.1	127.0.0.1	TCP	66	0.000103035	59368 → 8888 [FIN, ACK] Seq=363 A
22	0.021754659	127.0.0.1	127.0.0.1	TCP	66	0.000004091	8888 → 59368 [ACK] Seq=105 Ack=36
23	16.015325350	127.0.0.1	127.0.0.1	TCP	74	0.000000000	59370 → 8888 [SYN] Seq=0 Win=6549
24	16.015335928	127.0.0.1	127.0.0.1	TCP	74	0.000010578	8888 → 59370 [SYN, ACK] Seq=0 Ack=
25	16.015344882	127.0.0.1	127.0.0.1	TCP	66	0.000000954	59370 → 8888 [ACK] Seq=1 Ack=1 Wi
26	16.419346885	127.0.0.1	127.0.0.1	HTTP	442	0.404002003	GET /zenit.html HTTP/1.1
27	16.419359201	127.0.0.1	127.0.0.1	TCP	66	0.000012316	8888 → 59370 [ACK] Seq=1 Ack=377
28	16.419842811	127.0.0.1	127.0.0.1	TCP	251	0.000483610	8888 → 59370 [PSH, ACK] Seq=1 Ack=
29	16.419854001	127.0.0.1	127.0.0.1	TCP	66	0.000011190	59370 → 8888 [ACK] Seq=377 Ack=18
30	16.523752578	127.0.0.1	127.0.0.1	HTTP	308	0.103898577	HTTP/1.0 200 OK (text/html)
31	16.523761727	127.0.0.1	127.0.0.1	TCP	66	0.000009149	59370 → 8888 [ACK] Seq=377 Ack=42
32	16.523879238	127.0.0.1	127.0.0.1	TCP	66	0.000117511	8888 → 59370 [FIN, ACK] Seq=428 A
33	16.523958343	127.0.0.1	127.0.0.1	TCP	66	0.000079105	59370 → 8888 [FIN, ACK] Seq=377 A
34	16.523977070	127.0.0.1	127.0.0.1	TCP	66	0.000018727	8888 → 59370 [ACK] Seq=429 Ack=37
35	16.545011614	127.0.0.1	127.0.0.1	TCP	74	0.000000000	59372 → 8888 [SYN] Seq=0 Win=6549
36	16.545030191	127.0.0.1	127.0.0.1	TCP	74	0.000018577	8888 → 59372 [SYN, ACK] Seq=0 Ack=
37	16.545041358	127.0.0.1	127.0.0.1	TCP	66	0.000011167	59372 → 8888 [ACK] Seq=1 Ack=1 Wi
38	16.545130788	127.0.0.1	127.0.0.1	HTTP	361	0.000089430	GET /zenit.jpg HTTP/1.1
39	16.545134186	127.0.0.1	127.0.0.1	TCP	66	0.000003398	8888 → 59372 [ACK] Seq=1 Ack=296
40	16.545649269	127.0.0.1	127.0.0.1	TCP	254	0.000515083	8888 → 59372 [PSH, ACK] Seq=1 Ack=
41	16.545666259	127.0.0.1	127.0.0.1	TCP	66	0.000016990	59372 → 8888 [ACK] Seq=296 Ack=18
42	16.581273317	127.0.0.1	127.0.0.1	TCP	16450	0.035607058	8888 → 59372 [PSH, ACK] Seq=189 A
43	16.581297025	127.0.0.1	127.0.0.1	TCP	66	0.000023708	59372 → 8888 [ACK] Seq=296 Ack=16
44	16.581478429	127.0.0.1	127.0.0.1	TCP	16450	0.000181404	8888 → 59372 [PSH, ACK] Seq=16573
45	16.581483336	127.0.0.1	127.0.0.1	TCP	66	0.000004907	59372 → 8888 [ACK] Seq=296 Ack=32
46	16.582341804	127.0.0.1	127.0.0.1	TCP	16450	0.000085468	8888 → 59372 [PSH, ACK] Seq=32957
47	16.582350908	127.0.0.1	127.0.0.1	TCP	66	0.000009104	59372 → 8888 [ACK] Seq=296 Ack=49
48	16.582395421	127.0.0.1	127.0.0.1	HTTP	3225	0.000044513	HTTP/1.0 200 OK (PNG)
49	16.582398230	127.0.0.1	127.0.0.1	TCP	66	0.000002809	59372 → 8888 [ACK] Seq=296 Ack=52
50	16.582459392	127.0.0.1	127.0.0.1	TCP	66	0.000061162	8888 → 59372 [FIN, ACK] Seq=52500
51	16.582513311	127.0.0.1	127.0.0.1	TCP	66	0.000053919	59372 → 8888 [FIN, ACK] Seq=296 A
52	16.582518071	127.0.0.1	127.0.0.1	TCP	66	0.000004760	8888 → 59372 [ACK] Seq=52501 Ack=

Z danych wireshark-a najbardziej interesują nas komunikaty o protokole http. Możemy zauważyć pewien schemat. Po odświeżeniu strony lub po przejściu na podstronę przeglądarka wysyła zapytanie “GET” - zapytanie o konkretną podstronę. Gdy już otrzyma podstronę przeglądarka wysyła kolejne zapytanie o konkretne zasoby w tym wypadku są tą zdjęcia. Możemy to wywnioskować z kolumny info. W celu odczytania większej liczby szczegółowych informacji możemy klikając w odpowiedni komunikat rozszerzyć opcję **Hypertext Tranfser Protocol**. Wówczas możemy odczytać nagłówki zadanych żądań.