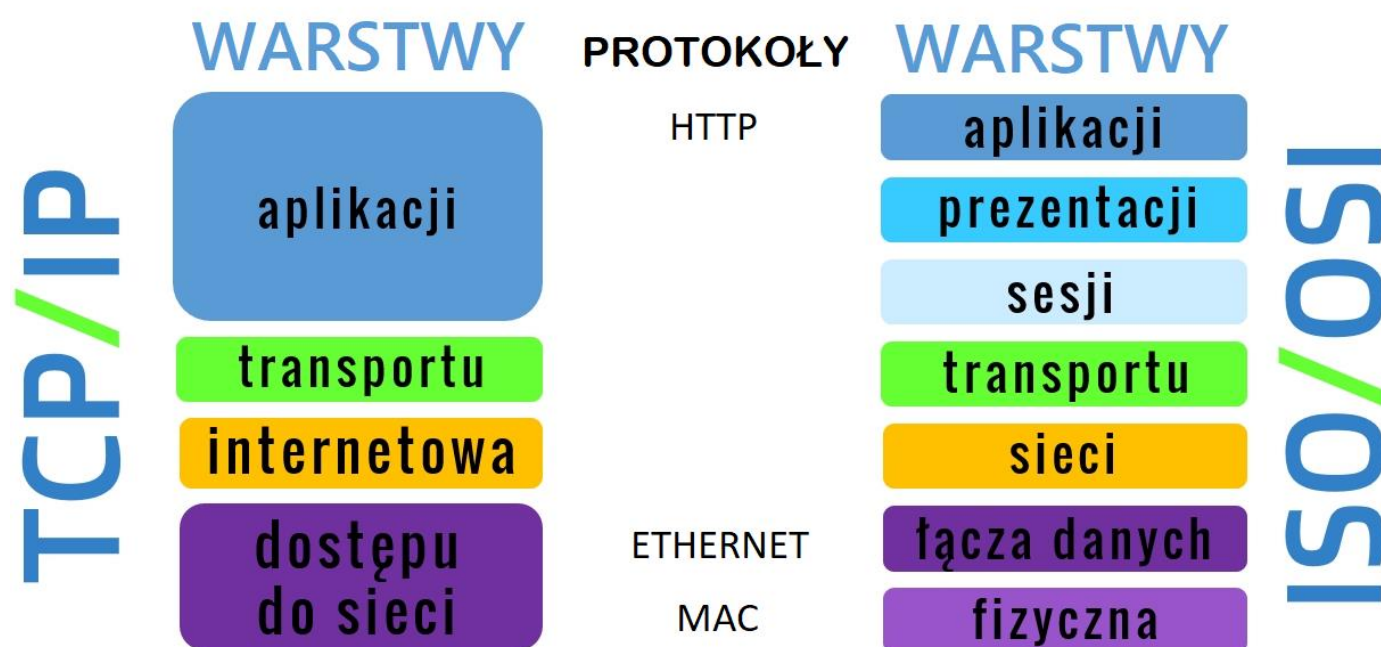


Sprawozdanie z Listy 5 (Technologie Sieciowe)

Jakub Omieljaniuk (250090)

W liście 3 omówiłem proces komunikacji między urządzeniami w sieci na najniższych warstwach modeli TCP/IP i ISO/OSI. Korzystaliśmy tam z protokołu Ethernet (ramkowanie danych) oraz protokołu CSMA/CD rozszerzającego protokół MAC (dostęp do medium transmisyjnego). Dane na tym poziomie były ciągami zer i jedynek, przekształcanymi na sygnał (prąd elektryczny lub fale), który jest rozprowadzany w medium transmisyjnym. Pokazałem również jak ciąg zer i jedynek jest zamieniany na czytelny dla programisty kod.

W liście 5 opiszę komunikację między urządzeniami (klientem a serwerem), odbywającą się na najwyższej warstwie modeli, czyli **warstwie aplikacji**. W tej warstwie operujemy na zrozumiałym dla użytkownika kodzie, korzystając m.in. z **protokołu HTTP – Hypertext Transfer Protocol**.



1. Warstwy modeli i wykorzystywane protokoły

Protokół HTTP określa zasady komunikacji między urządzeniami (programami) posługując się zdefiniowaną formą **żądań** wysyłanych do serwera i **odpowiedzi** odsyłanych do klienta. Przedmiotem żądania jest konkretny zasób, którym może być np. obrazek, strona HTML czy plik z kodem JavaScript. Każdy zasób jest identyfikowany poprzez nadany mu unikalny łańcuch znaków zwany **URI (Uniform Resource Identifier)**. Przykład URI:

/articles/sport/23hj42.html

Adres URL (Uniform Resource Locator) jest bardziej szczegółową formą identyfikacji zasobów, ponieważ musi zawierać dodatkowo ścieżkę do niego i podaje protokół, za pomocą którego można go ściągnąć:

http://www.sport.com/articles/sport/23hj42.html

Adres URL jest jednocześnie URI, natomiast nie każdy URI musi być adresem URL (tak samo jak każdy kwadrat jest prostokątem, ale nie każdy prostokąt jest kwadratem).

Interpretacja żądań serwera może być realizowana przez napisany program (skrypt) na serwerze. Przykład takiego programu został nam załączony do tej listy:

```
server3.pl X
1  use HTTP::Daemon;
2  use HTTP::Status;
3  #use IO::File;
4
5  my $d = HTTP::Daemon->new(
6      LocalAddr => 'localhost',
7      LocalPort => 4321,
8      ) || die;
9
10 print "Please contact me at: <URL:", $d->url, ">\n";
11
12
13 while (my $c = $d->accept) {
14     while (my $r = $c->get_request) {
15         if ($r->method eq 'GET') {
16
17             $file_s = "./index.html";    # index.html - jakiś istniejący plik
18             $c->send_file_response($file_s);
19
20         }
21         else {
22             $c->send_error(RC_FORBIDDEN)
23         }
24     }
25     $c->close;
26     undef($c);
27 }
28
29
```


2. Skrypt serwera obsługujący żądania klienta

W kodzie tym zmieniłem 6 linijkę, zamieniając lokalny adres na 'localhost' i odpaliłem go przez program Padre (Perl IDE). Stworzyłem również plik index.html w folderze, z którego odpalany jest skrypt server3.pl:

```
← → ↺ 🏠 ⓘ view-source:file:///F:/TechSieciowe/List5/index.html
1 <html>
2   <head>
3     <title>Technologie sieciowe</title>
4   </head>
5   <body>
6     <p>Hello World!</p>
7   </body>
8 </html>
```

3. Plik index.html

Do przeanalizowania wysyłanych informacji między klientem a serwerem posłużę się programem **curl** obsługiwany z linii komend. Flaga **-v** pozwala nam na uzyskanie szczegółowych informacji. Dane wysyłane przez klienta poprzedzone są znakiem **>**, natomiast odbierane - znakiem **<**:

 Wiersz polecenia

```
C:\Users\Jakbuczyk>curl -v http://localhost:4321/
* Trying ::1...
* TCP_NODELAY set
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 4321 (#0)
> GET / HTTP/1.1
> Host: localhost:4321
> User-Agent: curl/7.55.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 09 Jun 2020 01:04:41 GMT
< Server: libwww-perl-daemon/6.00
< Content-Type: text/html
< Content-Length: 139
< Last-Modified: Mon, 08 Jun 2020 21:51:30 GMT
<
<html>
  <head>
    <title>Technologie sieciowe</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>* Connection #0 to host localhost left intact
```

4. Wysłanie żądania do serwera

Poleceniem **curl -v http://localhost:4321/** nawiązałem połączenie z serwerem, poprzez wysłanie **żądania GET¹** zasobu **/**, czyli domyślnego (głównego) zasobu serwera, a także wysłałem informację o wersji protokołu HTTP, którym się komunikuję (HTTP/1.1). Ogólny format żądania zawiera:

- typ żądania¹, zasób do którego się odnosi i wersję protokołu,
- nagłówki
- pusta linia oznaczająca koniec nagłówków,
- ciało wiadomości (w przypadku żądań GET jest puste)

Od serwera otrzymałem najpierw **status odpowiedzi²** (200 OK), a także kod HTML domyślnej strony (w tym wypadku index.html). Ogólny format odpowiedzi zawiera:

- wersję protokołu i status odpowiedzi
- nagłówki
- pusta linia oznaczająca koniec nagłówków,
- ciało wiadomości (w tym przypadku kod HTML)

¹**czasownik HTTP** (typ żądania) – kluczowe słowo określające typ żądania. Najczęściej używane czasowniki to: **GET** (pobieranie zasobów z nagłówkami), **HEAD** (pobieranie jedynie nagłówków), **POST** (przesyłanie danych do serwera), **PUT** (aktualizacja danych na serwerze), **DELETE** (usuwanie danych z serwera).

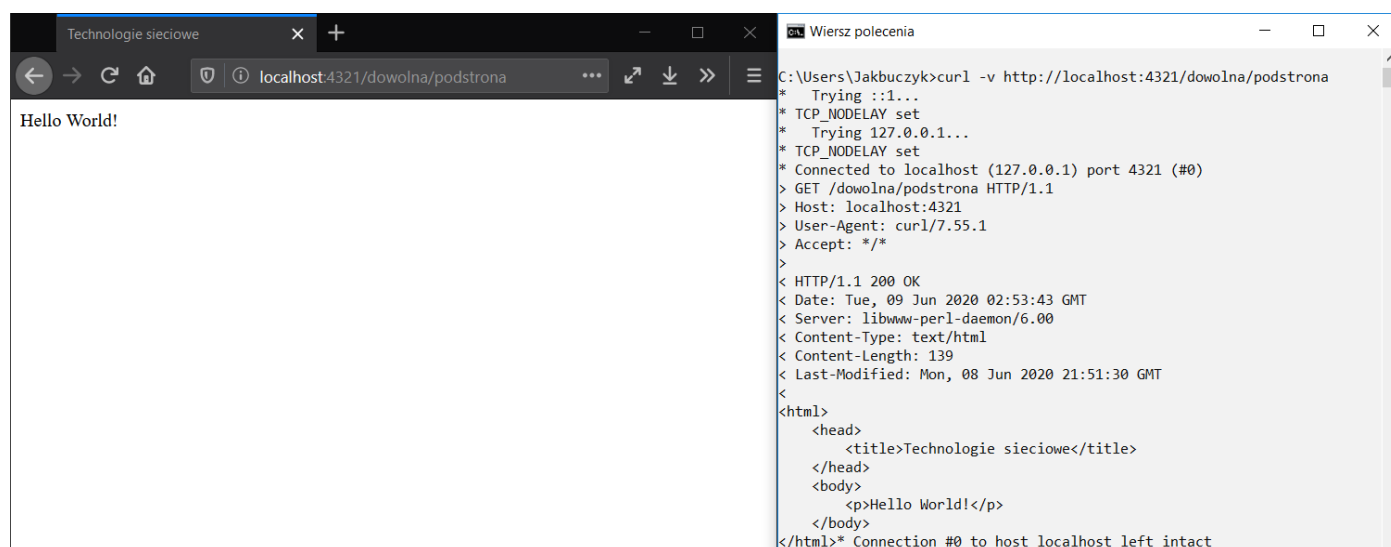
²**status odpowiedzi** – 3-cyfrowy kod z tytułem będący podstawową informacją czy dane żądanie się powiodło. Każdy status określa inną sytuację, np. **404 Not Found** mówi o tym, że żądanego zasób nie znaleziono na serwerze, a **200 OK**, że dane żądanie zostało poprawnie przetworzone. Dokładne opisy wszystkich statusów można znaleźć w Internecie.

Programem klienckim może być również przeglądarka internetowa, która samodzielnie wykona żądanie do serwera, podanego przez nas w adresie URL:



5. Strona serwera

Przy bardziej rozbudowanych stronach, przeglądarka może wysłać wiele żądań pobrania zasobów do serwera, aby załadować pojedynczą podstronę wraz ze wszystkimi obrazkami i skryptami. W naszym skrypcie, serwer czeka na dowolne żądanie typu GET (linijka 15 w server3.pl) i w odpowiedzi wysyła kod pliku index.html (linijka 18). Możemy zatem wpisać dowolną, nawet nieistniejącą, podstronę, a w odpowiedzi dostaniemy stronę index.html:



6. Strona serwera dla dowolnego zapytania GET

W przypadku zapytania innego niż GET (np. POST), otrzymamy w odpowiedzi status **403 Forbidden** (co jest zdefiniowane w naszym skrypcie server3.pl w 22 linijce):

```
Wiersz polecenia

C:\Users\Jakbuczyk>curl -v -X POST http://localhost:4321/dowolna/podstrona
* Trying ::1...
* TCP_NODELAY set
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 4321 (#0)
> POST /dowolna/podstrona HTTP/1.1
> Host: localhost:4321
> User-Agent: curl/7.55.1
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Date: Tue, 09 Jun 2020 02:55:16 GMT
< Server: libwww-perl-daemon/6.00
< Content-Type: text/html
< Content-Length: 53
<
<title>403 Forbidden</title>
<h1>403 Forbidden</h1>

* Connection #0 to host localhost left intact
```

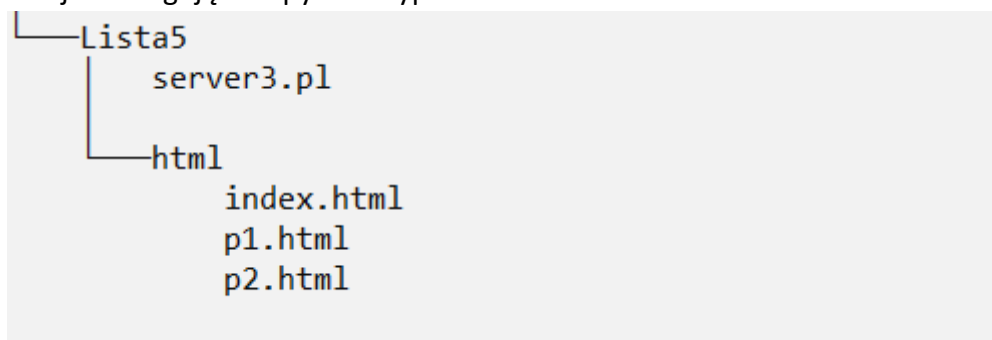
7. Żądanie POST wysłane do serwera

Aby serwer w odpowiedzi na zapytanie HEAD przesyłał nagłówek naszego zapytania, należy dodać obsługę zapytań HEAD w server3.pl w pętli while (linijki 21-26):

```
13 while (my $c = $d->accept) {
14     while (my $r = $c->get_request) {
15         if ($r->method eq 'GET') {
16
17             $file_s = "./index.html"; # index.html - jakiś istniejący plik
18             $c->send_file_response($file_s);
19
20         }
21         elif ($r->method eq 'HEAD') {
22
23             $response = $r->header('Accept');
24             $c->send_file_response($response);
25
26         }
27         else {
28             $c->send_error(RC_FORBIDDEN)
29         }
29     }
29 }
```

8. Dodanie obsługi żądania HEAD

W celu rozszerzenia funkcjonalności serwera do obsługi prostej struktury serwisu WWW, stworzyłem folder o nazwie *html* w katalogu, gdzie znajduje się skrypt. Zawarłem w nim 3 podstrony składające się na mój serwis: *index.html*, *p1.html*, *p2.html*. Następnie w skrypcie *server3.pl* należy zmienić instrukcje obsługujące zapytania typu GET:



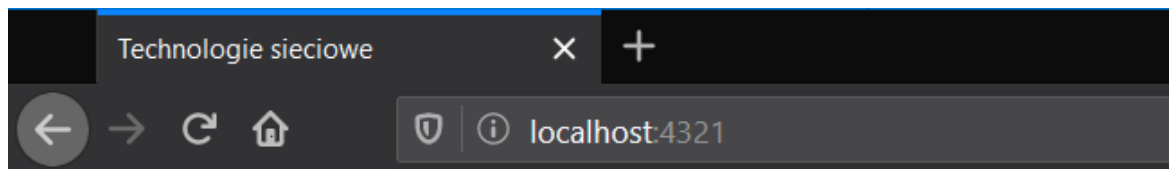
9. Struktura folderu serwera

```
index.html x p1.html x p2.html x
1  <html>
2    <head>
3      <title>Technologie sieciowe</title>
4    </head>
5    <body>
6      <h1>index.html</h1>
7      <p>Go to:</p>
8      <a href="http://localhost:4321/p1.html">p1.html</a><br>
9      <a href="http://localhost:4321/p2.html">p2.html</a>
10    </body>
11  </html>
```

10. Nowy kod *index.html* zawierający linki do pozostałych podstron

```
server3.pl x
13  while (my $c = $d->accept) {
14    while (my $r = $c->get_request) {
15      if ($r->method eq 'GET') {
16
17        my $uri = $r->uri;
18        if ($uri eq "/") {
19          $uri = "/index.html";
20        }
21        my $requested_file = "html" . $uri;
22
23        if ( -e $requested_file) {
24          $c->send_file_response($requested_file);
25        } else {
26          $c->send_error(RC_NOT_FOUND);
27        }
28      }
29    }
}
```

11. Zmodyfikowany fragment *server.pl*

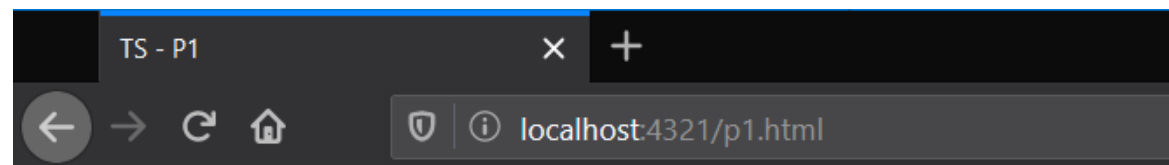


index.html

Go to:

[p1.html](#)

[p2.html](#)



p1.html

Go to:

[index.html](#)

[p2.html](#)

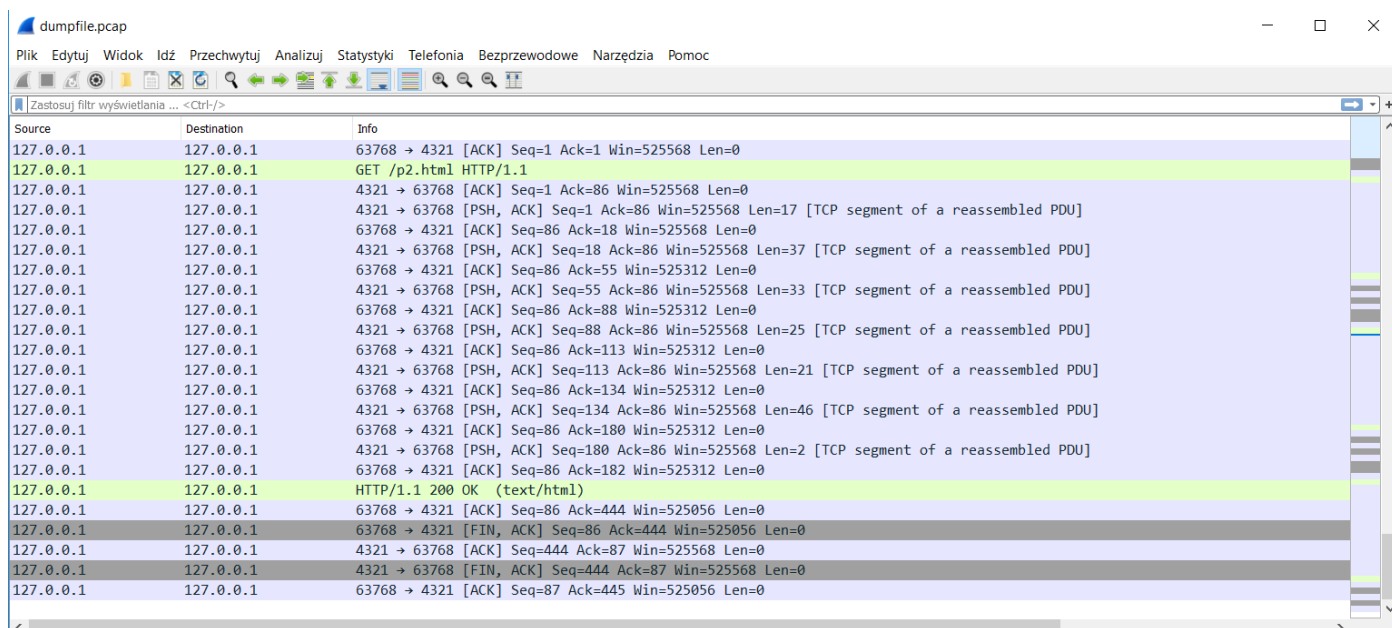
12. Serwis WWW w przeglądarce internetowej

Do przechwytywania komunikatów do/od serwera posłużę się programem **RawCap**:

```
C:\Users\Jakbuczyk\Downloads\RawCap.exe
Interfaces:
0.      169.254.208.164 Ethernet      Ethernet
1.      192.168.56.1   VirtualBox Host-Only Network  Ethernet
2.      169.254.164.52 Połączenie lokalne* 1   Wireless80211
3.      169.254.49.83  Połączenie lokalne* 2   Wireless80211
4.      169.254.60.195 ZeroTier One [e5cd7a9e1c04fd19] Ethernet
5.      192.168.0.100  VMware Network Adapter VMnet8 Ethernet
6.      169.254.209.227 VMware Network Adapter VMnet2 Ethernet
7.      169.254.2.14   VMware Network Adapter VMnet3 Ethernet
8.      169.254.123.27 VMware Network Adapter VMnet4 Ethernet
9.      169.254.65.223 VMware Network Adapter VMnet5 Ethernet
10.     192.168.0.45   Wi-Fi Wireless80211
11.     127.0.0.1      Loopback Pseudo-Interface 1 Loopback
Select interface to sniff [default '0']: 11
Output path or filename [default 'dumpfile.pcap']:
Sniffing IP : 127.0.0.1
Output File : C:\Users\Jakbuczyk\Downloads\dumpfile.pcap
--- Press [Ctrl]+C to stop ---
Packets      : 0
```

13. Włączenie przechwytywania komunikatów w programie RawCap

Historia przechwytywania komunikatów zapisze się w pliku `dumpfile.pcap`, który otworzę za pomocą programu **Wireshark**:



14. Analiza przechwycionych komunikatów w programie Wireshark

Na załączonym obrazku widać komunikaty związane z żądaniem wysłania strony `p2.html` przez klienta i pomyślnym jego przetworzeniem przez serwer. Pakiety z flagą ACK są używane do weryfikacji wysyłki i odbioru. Flaga FIN informuje o zakończeniu połączenia – po otrzymaniu pakietu z flagą FIN, program czeka na ostatni pakiet ACK i kończy połączenie. Flaga PSH wymusza określony priorytet przetworzenia pakietu.