Sprawozdanie
z laboratorium kursu
„Technologie sieciowe”

Lista 5 – „HTTP”

Bartosz Chodorowski <chomzee@ethernet.pl>
Numer indeksu: 166142

Wrocław, 25 maja 2009

1 Cele

Celem zajęć laboratoryjnych było przetestowanie prostego serwera HTTP napisanego w języku Perl oraz stworzenie podobnej, lecz nieco bardziej skomplikowanej aplikacji, za pomocą której możliwe będzie udostępnianie statycznego serwisu WWW.

2 Realizacja

Celem przetestowania skryptu `server3.pl` użyto programów `telnet`, *Mozilla Firefox* oraz `wireshark`.

Napisano również serwer HTTP o nazwie `kisshttpd`. Zaimplementowano w nim najważniejsze fragmenty protokołu HTTP[1] w wersji 1.0. Dodatkowo umieszczono w nim funkcjonalność, która wysyła do klienta nagłówek jego żądania. Użyto języka C.

2.1 `server3.pl` – zasada działania

Testowany skrypt `server3.pl` korzysta z modułu `HTTP::Daemon` do utworzenia serwera HTTP i odczytywania nadchodzących zapytań. Listing 1 przedstawia całą zawartość pliku.

Listing 1: Zawartość pliku `server3.pl`

```
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 - jakis istniejacy plik
18             $c->send_file_response($file_s);
19
20         }
21         else {
22             $c->send_error(RC_FORBIDDEN)
23         }
24
25     }
26     $c->close;
27     undef($c);
28 }
```

W liniach 5-8 następuje tworzenie nowego egzemplarza `$d` klasy `HTTP::Daemon`, co powoduje otwarcie do nasłuchu portu TCP 4321 dla połączeń przychodzących. Linia 10 powoduje wypisanie na standardowe wyjście

[1] `http://www.w3.org/Protocols/rfc1945/rfc1945`

informacji dotyczącej adresu URL, pod którym powinien znajdować się udostępniany zasób. W linii 13 w pętli tworzony jest egzemplarz `$c` klasy `HTTP::Daemon::ClientConn`, który umożliwiają komunikację z obecnie przetwarzanym, zaakceptowanym połączeniem. Pętla w liniach 14-25 przetwarza kolejne zapytania HTTP. Każde zapytanie metodą `GET` zostaje przetworzone w ten sam sposób – wysyłany jest plik `index.html` z bieżącego katalogu. Na zapytania metodą inną niż GET serwer odpowiada komunikatem *403 Forbidden*. W linii 26 serwer rozłącza klienta, po czym następuje powrót i obsługa kolejnego.

Głównym problemem związanym z powyższym skryptem jest brak wielowątkowości – wystarczy nawiązać i utrzymywać jedno połączenie z serwerem, aby wstrzymać jego pracę (akceptację kolejnych klientów).

2.2 `server3.pl` – przykładowe uruchomienia

Sposób w jaki można uruchomić rozważany program przedstawia listing 2. Sesję programu `telnet` przedstawia listing 3.

Listing 2: Uruchomienie skryptu `server3.pl`

```
1 chomzee@dropsik ~ $ echo 'Hello World!' > index.html
2 chomzee@dropsik ~ $ perl server3.pl
3 Please contact me at: <URL:http://localhost:4321/>
```

Listing 3: Połączenie z serwerem `server3.pl` za pomocą programu `telnet`

```
1 chomzee@dropsik ~ $ telnet localhost 4321
2 Trying 127.0.0.1...
3 Connected to localhost.
4 Escape character is '^]'.
5 GET / HTTP/1.0
6
7 HTTP/1.1 200 OK
8 Date: Sat, 23 May 2009 22:49:29 GMT
9 Server: libwww-perl-daemon/1.36
10 Content-Type: text/html
11 Content-Length: 13
12 Last-Modified: Sat, 23 May 2009 22:46:34 GMT
13
14 Hello World!
15 Connection closed by foreign host.
16 chomzee@dropsik ~ $
```

Linie 5-6 listingu 3 zostały wprowadzone z klawiatury. Obie linie zakończone są znakami <CR><LF> (powrót karetki, nowa linia). Linie 7-14 stanowią odpowiedź serwera – linia 7 jest odpowiedzią HTTP o kodzie 200 (OK). Pięć kolejnych linii stanowi nagłówki HTTP (między innymi aktualna data, oprogramowanie i wersja serwera, typ i długość przesyłanych danych oraz data ostatniej modyfikacji zasobu). Po pustej linii (13) następuje właściwa zawartość przesyłanego pliku złożona z jednej linii tekstu. Serwer natychmiast po przesłaniu danych zamyka połączenie, o czym `telnet` informuje nas linią `Connection closed by foreign host`.

Celem przetestowania serwera `server3.pl` użyto również programu *Mozilla Firefox*, otwierając adres URL `http://localhost:4321/`, po czym za pomocą opcji *Follow TCP Stream* programu `wireshark` podsłuchano właściwe połączenie TCP. Pełną komunikację obrazuje listing 4.

Listing 4: Komunikacja pomiędzy programem *Mozilla Firefox*, a skryptem `server2.pl`

```
GET / HTTP/1.1
Host: localhost:4321
User-Agent: Mozilla/5.0 (X11; U; Linux i686; pl; rv:1.9.0.5) Gecko/2008122117
    Gentoo Firefox/3.0.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pl,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
If-Modified-Since: Sat, 23 May 2009 21:17:34 GMT

HTTP/1.1 200 OK
Date: Sun, 24 May 2009 10:28:40 GMT
Server: libwww-perl-daemon/1.36
Content-Type: text/html
Content-Length: 13
Last-Modified: Sat, 23 May 2009 22:46:34 GMT

Hello World!
GET /favicon.ico HTTP/1.1
Host: localhost:4321
User-Agent: Mozilla/5.0 (X11; U; Linux i686; pl; rv:1.9.0.5) Gecko/2008122117
    Gentoo Firefox/3.0.5
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: pl,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

HTTP/1.1 200 OK
Date: Sun, 24 May 2009 10:28:40 GMT
Server: libwww-perl-daemon/1.36
Content-Type: text/html
Content-Length: 13
Last-Modified: Sat, 23 May 2009 22:46:34 GMT

Hello World!
GET /favicon.ico HTTP/1.1
Host: localhost:4321
User-Agent: Mozilla/5.0 (X11; U; Linux i686; pl; rv:1.9.0.5) Gecko/2008122117
    Gentoo Firefox/3.0.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pl,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

HTTP/1.1 200 OK
Date: Sun, 24 May 2009 10:28:43 GMT
Server: libwww-perl-daemon/1.36
Content-Type: text/html
Content-Length: 13
Last-Modified: Sat, 23 May 2009 22:46:34 GMT

Hello World!
```

Wiersze 1-11, 20-29 oraz 38-47 listingu 4 zawierają żądania HTTP wysłane przez przeglądarkę internetową. Pozostałe dane są odpowiedzią serwera. Wszystkie zapytania są zgodne z wersją 1.1 protokołu HTTP[2].

Pierwsze z zapytań dotyczy głównego pliku. Klient korzysta z metody `GET` oraz wysyła mnogość nagłówków nieświadomy tego, że prosty serwer odbierający je po drugiej stronie i tak ignoruje większość z nich. Serwer odpowiada zgodnie z protokołem.

Drugie zapytanie dotyczy pliku `/favicon.ico`. Wiele z nowoczesnych przeglądarek internetowych, otwierając główną stronę na danym serwerze, próbuje pobrać plik o takiej nazwie, aby wyświetlić małą ikonę obok paska adresu (o ile plik HTML nie definiuje explicite innego pliku ikony). Serwer jednak nie rozpoznaje intencji przeglądarki, gdyż na każde zapytanie metodą `GET` odpowiada w ten sam sposób – wysyła plik typu `text/html` o treści „Hello World!".

Jako że w drugim zapytaniu klient zaznaczył, iż spodziewa się raczej obrazka (uczynił to za pomocą nagłówka `Accept` w linii 23), ponawia żądanie dotyczące pliku `/favicon.ico`, zaznaczając tym razem, że zadowoli się odpowiedzią typu `text/html` (linia 41).

Wszystkie zapytania odbywają się wewnątrz jednego połączenia TCP, gdyż pozwala na to protokół HTTP/1.1 (regulują to nagłówki `Connection` oraz `Keep-Alive`).

Program *Mozilla Firefox* wyświetla zawartość pliku `index.html` – tekst „Hello World!".

2.3 `kisshttpd` – zasada działania

Sposób działania programu `kisshttpd` najlepiej obrazuje pseudokod zawarty w listingu 5.

Listing 5: `kisshttpd` – pseudokod

```
1  Utwórz gniazdo, przypisz mu port TCP 80
2  Zmień katalog bieżący na katalog domowy użytkownika kisshttpd
3  Uczyń bieżący katalog katalogiem głównym
4  Zrzuć uprawnienia na kisshttpd:kisshttpd
5  Akceptuj oczekującego klienta
6  Jeśli ilość klientów jest równa MAX_CLIENTS, odrzuć klienta i wróć do linii 5
7  Zwiększ ilość klientów o jeden
8  Utwórz nowy proces - wywołaj fork()
9  W procesie matce wróć do linii 5
10 Obsłuż klienta
11 Zmniejsz ilość klientów o jeden
12 Zakończ proces potomny
```

Czynności 1-4 z powyższego listingu wykonywane są celem zapewnienia bezpieczeństwa systemu, w którym serwer jest uruchamiany. Aby otworzyć port 80, proces potrzebuje uprawnień administratora, zatem program musi być uruchomiony przez użytkownika `root`. Jednak zaraz po utworzeniu gniazda proces zamyka się w „więzieniu" (ang. *jail*) za pomocą wywołania `chroot()`. Po zrzuceniu uprawnień na grupę i użytkownika `kisshttpd`, serwer nie ma możliwości dostępu do plików, które mają ustawione restrykcyjne prawa dostępu, jak również innych niż te zawarte w katalogu domowym użytkownika `kisshttpd`. Zabieg ten powoduje, że nawet w przypadku przejęcia kontroli nad programem przez potencjalnego włamywacza, bezpieczeństwo systemu jest zagrożone w minimalnym stopniu.

Dalsze czynności nie odbiegają od utartego schematu według jakiego postępuje wiele serwerów usług świadczonych za pomocą gniazd strumieniowych. Dodatkowo zliczamy każdego z klientów. Po zaakceptowaniu każdego kolejnego połączenia sprawdzany, czy jest jeszcze dla niego miejsce (tak, aby ilość aktualnie obsługiwanych klientów nie przekroczyła z góry ustalonej wartości `MAX_CLIENTS`).

[2] `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

Obsługa klientów odbywa się równolegle za pomocą duplikowania bieżącego procesu wywołaniem systemowym `fork()`. Po zakończeniu działania każdego z procesów potomnych, proces-matka reaguje na sygnał `SIGCHLD` usuwając z systemu „proces zombie" oraz zmniejszając licznik obsługiwanych klientów.

Zaraz po wywołaniu `fork()` nowy proces potomny duplikuje deskryptor gniazda na deskryptory standardowego wejścia, wyjścia i błędu. Właściwy dialog i obsługa protokołu HTTP zaimplementowano w innym pliku źródłowym – `client.c` w funkcji `serveClient()`.

Zaimplementowano obsługę metod GET oraz HEAD – przy zapytaniu metodą POST serwer zwróci błąd *501 Not Implemented*. Serwer w najprostszy możliwy sposób odpowiada na zapytania udostępniając pliki o zadanym URI.

Dodatkową funkcją jest specyficzna odpowiedź na zapytanie o URI `/header`. Serwer zwraca wówczas odpowiedź *200 OK*, wysyłając jako treść zapytanie nadesłane przez klienta wraz z nagłówkami HTTP.

2.4 `kisshttpd` – przykładowe uruchomienia

Po uruchomieniu programu `kisshttpd` przez użutkownika `root`, port TCP/80 zostaje otwarty do nasłuchu. Przykładową sesję programu `telnet` prezentującą zapytanie i odpowiedź przedstawia listing 6.

Listing 6: Program `telnet` wysyłający żadanie i odbierający odpowiedź od serwera `kisshttpd`

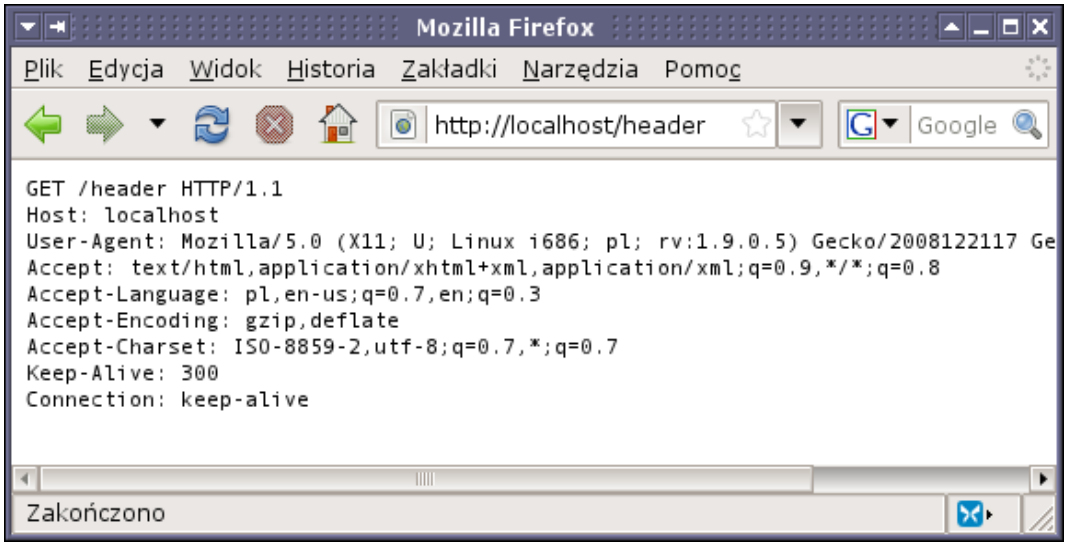
```
1  chomzee@dropsik ~ $ telnet localhost 80
2  Trying 127.0.0.1...
3  Connected to localhost.
4  Escape character is '^]'.
5  GET / HTTP/1.0
6
7  HTTP/1.0 200 OK
8  Server: kisshttpd/0.1
9
10 <html>
11         <body>
12         <h1>It works!</h1>
13
14         <hr>
15
16         <ul>
17                 <li><a href="first.html">First link</a>
18                 <li><a href="second.html">Second link</a>
19                 <li><a href="third.html">Third link</a>
20         </ul>
21
22         <hr>
23
24         <a href="/header">Chceck your HTTP request and headers</a>
25
26         </body>
27 </html>
28 Connection closed by foreign host.
29 chomzee@dropsik ~ $
```

Linie 5-6 stanowią najprostsze możliwe zapytanie HTTP/1.0. Linie 7-9 są odpowiedzią HTTP zawierającą tylko jeden nagłówek – `Server`. W liniach 10-27 znajduje się zawartość transmitowanego pliku (`index.html`).

Programem *Mozilla Firefox*, jak również wieloma innymi powszechnie dostępnymi przeglądarkami internetowymi, można wygodnie przeglądać zasoby udostępniane przez program `kisshttpd`. Rysunek 1 przedstawia

zrzutkę z ekranu programu `Mozilla Firefox` prezentującą funkcję odsyłania żądania HTTP wraz z nagłówkami.

Rysunek 1: `http://localhost/header` w przeglądarce *Mozilla Firefox*



3 Wnioski

Protokół HTTP jest względnie prostym, czytelnym, tekstowym protokołem pracującym w warstwie aplikacji. Jego implementacja (zwłaszcza serwera, który nie musi nic wyświetlać, tylko zarządzać zasobami) nie jest trudna. Korzystając z języków programowania wysokiego poziomu i odpowiednich bibliotek (np. Perl z `HTTP::Daemon`) można szybko rozbudować prosty serwer WWW, implementując w nim poszczególne funkcjonalności. Jednak implementacja całego protokołu HTTP w wersji 1.1 od zera z pewnością okazałaby się dość pracochłonna.