

# Programowanie aplikacji sieciowych

Zbiór zadań, część pierwsza

Katarzyna Mazur

13 lipca 2023

## Spis treści

<b>1</b>	<b>Zadania wprowadzające</b>	<b>3</b>
<b>2</b>	<b>Analiza pakietów sieciowych</b>	<b>4</b>
<b>3</b>	<b>Gniazda klienckie</b>	<b>6</b>
<b>4</b>	<b>Gniazda serwerowe</b>	<b>12</b>
<b>5</b>	<b>Bezpiecznie gniazda</b>	<b>19</b>
<b>6</b>	<b>Protokoły pocztowe</b>	<b>20</b>
<b>7</b>	<b>Protokół HTTP (w wersji 1.1)</b>	<b>24</b>
<b>8</b>	<b>Odpowiedzi</b>	<b>33</b>

## 1 Zadania wprowadzające

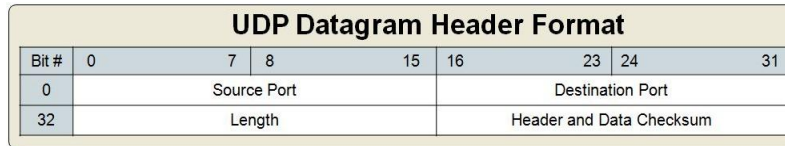
- 1.1 Napisz program, w którym pobierzesz od użytkownika nazwę pliku tekstowego w formacie `*.txt`, a następnie skopiujesz go do pliku pod nazwą `lab1zad1.txt`. Zadbaj o prawidłową obsługę błędów.
- 1.2 Napisz program, w którym pobierzesz od użytkownika nazwę pliku graficznego w formacie `*.png`, a następnie skopiujesz go do pliku pod nazwą `lab1zad2.png`. Zadbaj o prawidłową obsługę błędów.
- 1.3 Napisz program, w którym pobierzesz od użytkownika adres IPv4, a następnie sprawdzisz, czy jest on prawidłowym adresem. Zadbaj o prawidłową obsługę błędów. *Podpowiedź: zadanie możesz rozwiązać przy pomocy wyrażeń regularnych*
- 1.4 Napisz program, w którym pobierzesz od użytkownika adres IPv6, a następnie sprawdzisz, czy jest on prawidłowym adresem. Zadbaj o prawidłową obsługę błędów. *Podpowiedź: zadanie możesz rozwiązać przy pomocy wyrażeń regularnych*
- 1.5 Napisz program, który jako argument linii poleceń pobierze od użytkownika adres IPv4, a następnie wyświetli odpowiadającą mu nazwę `hostname` (nazwę domenową). Zadanie rozwiąż bez użycia dodatkowych bibliotek. Zadbaj o prawidłową obsługę błędów.

## 2 Analiza pakietów sieciowych

**2.1** Poniżej znajduje się pełny zapis datagramu UDP w postaci szesnastkowej.

```
ed 74 0b 55 00 24 ef fd 70 72 6f 67 72 61
6d 6d 69 6e 67 20 69 6e 20 70 79 74 68 6f
6e 20 69 73 20 66 75 6e
```

Wiedząc, że w zapisie szesnastkowym jedna cyfra reprezentuje 4 bity, oraz znając strukturę datagramu UDP:



Napisz program, który z powyższego datagramu UDP wydobędzie:

- numer źródłowego portu
- numer docelowego portu
- dane (ile bajtów w tym pakiecie zajmują dane?)

A następnie uzyskany wynik w postaci: `zad2.1odp;src;X;dst;Y;data;Z` gdzie:

- X to wydobyty z pakietu numer portu źródłowego
- Y to wydobyty z pakietu numer portu docelowego
- Z to wydobyte z pakietu dane

prześle do serwera UDP działającego na wskazanym porcie pod podanym adresem IPv4, w celu sprawdzenia, czy udało się prawidłowo odczytać wymagane pola. Serwer zwróci odpowiedź TAK lub NIE, a w przypadku błędnego sformatowania wiadomości, odeśle odpowiedź BAD\_SYNTAX. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**2.2** Zmodyfikuj program z zadania **2.1** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**2.3** Poniżej znajduje się pełny zapis segmentu TCP w postaci szesnastkowej (pole opcji ma 12 bajtów).

```
0b 54 89 8b 1f 9a 18 ec bb b1 64 f2 80 18
00 e3 67 71 00 00 01 01 08 0a 02 c1 a4 ee
00 1a 4c ee 68 65 6c 6c 6f 20 3a 29
```

Wiedząc, że w zapisie szesnastkowym jedna cyfra reprezentuje 4 bity, oraz znając strukturę segmentu TCP:

TCP Segment Header Format								
Bit #	0	7	8	15	16	23	24	31
0	Source Port				Destination Port			
32	Sequence Number							
64	Acknowledgment Number							
96	Data Offset	Res	Flags		Window Size			
128	Header and Data Checksum				Urgent Pointer			
160...	Options							

Napisz program, który z powyższego segmentu TCP wydobędzie:

- numer źródłowego portu
- numer docelowego portu
- dane (ile bajtów w tym pakiecie zajmują dane?)

A następnie uzyskany wynik w postaci: `zad2.3odp;src;X;dst;Y;data;Z` gdzie:

- X to wydobyty z pakietu numer portu źródłowego
- Y to wydobyty z pakietu numer portu docelowego
- Z to wydobyte z pakietu dane

prześle do serwera TCP działającego na wskazanym porcie pod podanym adresem IPv4, w celu sprawdzenia, czy udało się prawidłowo odczytać wymagane pola. Serwer zwróci odpowiedź **TAK** lub **NIE**, a w przypadku błędnego sformatowania wiadomości, odeśle odpowiedź **BAD\_SYNTAX**. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

- 2.4** Zmodyfikuj program z zadania **2.3** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

### 3 Gniazda klienckie

## Gniazda w języku Python - moduł socket

#### Tworzenie gniazd klienckich/serwerowych TCP, IPv4

```
#!/usr/bin/env python3
import socket

if __name__ == '__main__':

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockIPv4.close()
```

#### Tworzenie gniazd klienckich/serwerowych TCP, IPv6

```
#!/usr/bin/env python3
import socket

if __name__ == '__main__':

    sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
    sockIPv6.close()
```

#### Tworzenie gniazd klienckich/serwerowych UDP, IPv4

```
#!/usr/bin/env python3
import socket

if __name__ == '__main__':

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockIPv4.close()
```

#### Tworzenie gniazd klienckich/serwerowych UDP, IPv6

```
#!/usr/bin/env python3
import socket

if __name__ == '__main__':

    sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
    sockIPv6.close()
```

**Gniazdo klienckie TCP, IPv4: nawiązanie połączenia z serwerem**

```
#!/usr/bin/env python3
import socket

if __name__ == "__main__":

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockIPv4.settimeout(5)

    try:
        sockIPv4.connect(("127.0.0.1", 80))
    except socket.error as exc:
        print(f"Wyjatek socket.error: {exc}")

    sockIPv4.close()
```

**Gniazdo klienckie TCP, IPv6: nawiązanie połączenia z serwerem**

```
#!/usr/bin/env python3
import socket

if __name__ == "__main__":

    address = socket.getaddrinfo("::1", 80, socket.AF_INET6)
    sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
    sockIPv6.settimeout(5)

    try:
        sockIPv6.connect(address[0][4])
    except socket.error as exc:
        print(f"Wyjatek socket.error: {exc}")

    sockIPv6.close()
```

**Gniazdo klienckie TCP, IPv4: komunikacja z serwerem (wysyłanie i odbieranie danych)**

```
#!/usr/bin/env python3
import socket

if __name__ == "__main__":

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockIPv4.settimeout(5)

    try:
        sockIPv4.connect(("127.0.0.1", 80))
        sockIPv4.sendall("Hello Server!".encode()) # wysyłanie
        print(sockIPv4.recv(1024).decode())         # odbieranie
    except socket.error as exc:
        print(f"Wyjatek socket.error: {exc}")

    sockIPv4.close()
```

**Gniazdo klienckie TCP, IPv6: komunikacja z serwerem (wysyłanie i odbieranie danych)**

```
#!/usr/bin/env python3
import socket

if __name__ == "__main__":

    address = socket.getaddrinfo("::1", 80, socket.AF_INET6)
    sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
    sockIPv6.settimeout(5)

    try:
        sockIPv6.connect(address[0][4])
        sockIPv6.sendall("Hello Server!".encode()) # wysyłanie
        print(sockIPv6.recv(1024).decode())         # odbieranie
    except socket.error as exc:
        print(f"Wyjatek socket.error: {exc}")

    sockIPv6.close()
```

**Gniazdo klienckie UDP, IPv4: komunikacja z serwerem (wysyłanie i odbieranie danych)**

```
#!/usr/bin/env python3

import socket

HOST = '127.0.0.1'
PORT = 80

sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = (HOST, PORT)

try:
    message = "Hello Server!"
    sent = sockIPv4.sendto(message.encode(), server_address) # wysyłanie
    data, server = sockIPv4.recvfrom(4096)                  # odbieranie
    print(f"Received: {data.decode()}")

finally:
    sockIPv4.close()
```

**Gniazdo klienckie UDP, IPv6: komunikacja z serwerem (wysyłanie i odbieranie danych)**

```
#!/usr/bin/env python3

import socket

HOST = "::1"
PORT = 80

sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)

try:
    sockIPv6.sendto("Hello Server!".encode(), (HOST, PORT)) # wysyłanie
    data, server = sockIPv6.recvfrom(4096)                  # odbieranie
    print(f"Received: {data.decode()}")

finally:
    sockIPv6.close()
```



## Gniazda TCP

- 3.1** Napisz program klienta, w którym połączysz się z serwerem na danym porcie przy użyciu protokołu TCP. Adres IPv4 serwera i numer portu pobierz jako argumenty linii poleceń. Wyświetl informację, czy udało się nawiązać połączenie. Program powinien akceptować adres w postaci adresu IPv4 jak i hostname. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.2** Zmodyfikuj program z zadania **3.1** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.3** Napisz program klienta (prosty skaner portów sieciowych), który dla danego serwera przy użyciu protokołu TCP będzie sprawdzał, jakie porty są otwarte, a jakie zamknięte. Adres IPv4 serwera pobierz jako argument linii poleceń. Program powinien akceptować adres w postaci adresu IPv4 jak i hostname. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.4** Zmodyfikuj program z zadania **3.3** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.5** Napisz program klienta, który z serwera o podanym adresie IPv4 i porcie pobierze aktualną datę i czas, a następnie wyświetli je na konsoli. Adres IPv4 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.6** Zmodyfikuj program z zadania **3.5** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.7** Napisz program klienta, który połączy się z serwerem TCP działającym pod podanym adresem IPv4 na podanym porcie, a następnie wyśle do niego wiadomość i odbierze odpowiedź. Adres IPv4 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.8** Zmodyfikuj program z zadania **3.7** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.9** Napisz program klienta, który połączy się z serwerem TCP działającym pod podanym adresem IPv4 na podanym porcie, a następnie będzie w pętli wysyłał do niego tekst wczytany od użytkownika (jako argument wywołania programu bądź jako dane podawane na konsoli), i odbierał odpowiedzi. Adres IPv4 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.10** Zmodyfikuj program z zadania **3.9** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.11** Napisz program klienta (prosty skaner portów sieciowych), który dla danego serwera przy użyciu protokołu TCP będzie sprawdzał, jakie porty są otwarte, a jakie zamknięte. Oprócz informacji o otwartych / zamkniętych portach, program powinien również wyświetlać informację o tym, jaka usługa jest uruchomiona na danym porcie (baner usługi). Adres IPv4 serwera pobierz jako argument linii poleceń. Program

powinien akceptować adres w postaci adresu IPv4 jak i hostname. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**3.12** Zmodyfikuj program z zadania **3.11** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**3.13** Napisz program klienta, który połączy się z serwerem TCP działającym pod podanym adresem IPv4 na podanym porcie, a następnie wyśle do niego wiadomość i odbierze odpowiedź. Warunkiem zadania jest, aby klient wysłał i odebrał od serwera wiadomość o maksymalnej długości 20 znaków. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów. Uwzględnij sytuacje, gdy:

- wiadomość do wysłania jest za krótka - ma być wówczas uzupełniana do 20 znaków znakami spacji
- wiadomość do wysłania jest za długa - ma być przycięta do 20 znaków (lub wysłana w całości - sprawdź, co się wówczas stanie)

**3.14** Zmodyfikuj program z zadania **3.13** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**3.15** W implementacji gniazd, funkcje wysyłania i odbierania danych (np. funkcje `recv` i `send` w Pythonie) nie gwarantują wysłania / odbioru wszystkich danych. Rozważmy funkcję `recv`. Przykładowo, 100 bajtów może zostać wysłane jako grupa po 10 bajtów, albo od razu w całości. Oznacza to, iż jeśli używamy gniazd TCP, musimy odbierać dane, dopóki nie mamy pewności, że odebraliśmy odpowiednią ich ilość.

Napisz program klienta, który połączy się z serwerem TCP działającym pod podanym adresem IPv4 na podanym porcie, a następnie wyśle do niego wiadomość i odbierze odpowiedź. Dane odbieraj / wysyłaj w ten sposób, aby mieć pewność, że klient w rzeczywistości odebrał / wysłał wiadomość o wymaganej długości. Prawidłowa komunikacja powinna odbywać się w następujący sposób:

- Klient wysła dane do serwera
- Serwer odsyła klientowi odebrane od niego dane (identyczną wiadomość)

**Aby ułatwić sobie zadanie wysyłania i odbierania CAŁEJ wiadomości, możesz każdą z wysłanych / odebranych wiadomości kończyć znakiem nowej linii - `\n`.** Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Numer portu TCP, na którym ma działać serwer, pobierz z linii poleceń. Zadbaj o prawidłową obsługę błędów.

**3.16** Zmodyfikuj program z zadania **3.15** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

## Gniazda UDP

**3.17** Napisz program klienta, który połączy się z serwerem UDP działającym pod podanym adresem IPv4 na podanym porcie, a następnie wyśle do niego wiadomość i odbierze odpowiedź. Adres IPv4 serwera oraz numer portu pobierz jako argumenty linii poleceń. Program powinien akceptować adres w postaci adresu IPv4 jak i hostname. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**3.18** Zmodyfikuj program z zadania **3.17** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**3.19** Napisz program klienta, który połączy się z serwerem UDP działającym pod podanym adresem IPv4 na podanym porcie, a następnie będzie w pętli wysyłał do niego tekst wczytany od użytkownika, i odbierał odpowiedzi. Adres IPv4 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania

kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

- 3.20** Zmodyfikuj program z zadania **3.19** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.21** Napisz program klienta, który połączy się z serwerem UDP działającym pod podanym adresem IPv4 na podanym porcie, a następnie prześle do serwera liczbę, operator, liczbę (pobrane od użytkownika) i odbierze odpowiedź. Adres IPv4 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.22** Zmodyfikuj program z zadania **3.21** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.23** Napisz program klienta, który połączy się z serwerem UDP działającym pod podanym adresem IPv4 na podanym porcie, a następnie prześle do serwera pobrany z linii poleceń adres IP, i odbierze odpowiadającą mu nazwę hostname. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.24** Zmodyfikuj program z zadania **3.23** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.25** Napisz program klienta, który połączy się z serwerem UDP działającym pod podanym adresem IPv4 na podanym porcie, a następnie prześle do serwera nazwę hostname pobraną z linii poleceń, i odbierze odpowiadający mu adres IP. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 3.26** Zmodyfikuj program z zadania **3.25** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Adres IPv6 serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

## 4 Gniazda serwerowe

### Gniazda w języku Python - moduł socket

#### Tworzenie gniazd serwerowych TCP, IPv4

```
#!/usr/bin/env python3
import socket

if __name__ == '__main__':

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockIPv4.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sockIPv4.close()
```

#### Tworzenie gniazd serwerowych TCP, IPv6

```
#!/usr/bin/env python3
import socket

if __name__ == '__main__':

    sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
    sockIPv6.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sockIPv6.close()
```

#### Tworzenie gniazd serwerowych UDP, IPv4

```
#!/usr/bin/env python3
import socket

if __name__ == '__main__':

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockIPv4.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sockIPv4.close()
```

#### Tworzenie gniazd serwerowych UDP, IPv6

```
#!/usr/bin/env python3
import socket

if __name__ == '__main__':

    sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
    sockIPv6.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sockIPv6.close()
```

**Gniazdo serwerowe TCP, IPv4: nasłuchiwanie na danym porcie**

```
#!/usr/bin/env python3
import socket

if __name__ == "__main__":

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockIPv4.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    server_address = ('127.0.0.1', 80)
    sockIPv4.bind(server_address)
    sockIPv4.listen(1)

    sockIPv4.close()
```

**Gniazdo serwerowe TCP, IPv6: nasłuchiwanie na danym porcie**

```
#!/usr/bin/env python3
import socket

if __name__ == "__main__":

    sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
    sockIPv6.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    sockIPv6.bind(":::1", 80)
    sockIPv6.listen(1)

    sockIPv6.close()
```

**Gniazdo serwerowe TCP, IPv4: komunikacja z klientem (wysyłanie i odbieranie danych)**

```
#!/usr/bin/env python3
import socket

if __name__ == "__main__":

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockIPv4.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    server_address = ("127.0.0.1", 80)
    sockIPv4.bind(server_address)
    sockIPv4.listen(1)

    connection, client_address = sockIPv4.accept()

    try:
        data = connection.recv(1024).decode()
        if data:
            connection.sendall(data.encode())
            print(data)
    finally:
        connection.close()
```

**Gniazdo serwerowe TCP, IPv6: komunikacja z klientem (wysyłanie i odbieranie danych)**

```
#!/usr/bin/env python3
import socket

HOST = ':::1' # localhost
PORT = 80

def main():
    with socket.socket(socket.AF_INET6, socket.SOCK_STREAM) as server_socket:
        server_socket.bind((HOST, PORT))
        server_socket.listen(1)
        print(f'Server listening on {HOST}:{PORT}')

        client_socket, client_addr = server_socket.accept()
        print(f'Connected to client {client_addr[0]}:{client_addr[1]}')

        data = client_socket.recv(1024)    # odbieranie
        if data:
            print(f'Received data: {data.decode()}')
            client_socket.sendall(data)     # wysyłanie

        client_socket.close()
        print(f'Connection closed with client {client_addr[0]}:{client_addr[1]}')

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

**Gniazdo serwerowe UDP, IPv4: komunikacja z klientem (wysyłanie i odbieranie danych)**

```
#!/usr/bin/env python3
import socket

if __name__ == "__main__":
    host = "127.0.0.1"
    port = 80

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind((host, port))

    print(f"UDP Echo Server is listening on {host}:{port}...")

    data, address = server_socket.recvfrom(1024)    # odbieranie
    print(f"Received data from {address[0]}:{address[1]}: {data.decode()}")

    server_socket.sendto(data, address)            # wysyłanie
    print(f"Sent data back to {address[0]}:{address[1]}: {data.decode()}")

    server_socket.close()
```

**Gniazdo serwerowe UDP, IPv6: komunikacja z klientem (wysyłanie i odbieranie danych)**

```
#!/usr/bin/env python3

import socket

HOST = ":::1"
PORT = 80

sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
sockIPv6.close()
```

**Uwaga:** W poniższych zadaniach zakładamy, iż serwer powinien obsługiwać tylko jednego klienta w danej chwili.

## Gniazda TCP

**4.1** Napisz program serwera, który działając pod adresem IPv4 oraz na podanym porcie TCP, dla podłączającego się klienta, będzie odsyłał mu aktualny czas oraz datę. Prawidłowa komunikacja powinna odbywać się w następujący sposób:

- Serwer odbiera od klienta wiadomość (dowolną)
- Serwer odsyła klientowi aktualną datę i czas

Po zakończeniu obsługi klienta, serwer powinien nieprzerwanie oczekiwać na kolejnych klientów, których będzie mógł obsłużyć. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Numer portu TCP, na którym ma działać serwer, pobierz z linii poleceń. Zadbaj o prawidłową obsługę błędów.

**4.2** Zmodyfikuj program z zadania 4.1 w taki sposób, aby serwer działał pod adresem IPv6. Po zakończeniu obsługi klienta, serwer powinien nieprzerwanie oczekiwać na kolejnych klientów, których będzie mógł obsłużyć. Numer portu TCP pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**4.3** Napisz program serwera, który działając pod adresem IPv4 oraz na podanym porcie TCP, dla podłączającego się klienta, będzie odsyłał mu przesłaną wiadomość (tzw. serwer echa). Prawidłowa komunikacja powinna odbywać się w następujący sposób:

- Serwer odbiera dane od klienta
- Serwer odsyła klientowi odebrane od niego dane (identyczną wiadomość)

Po zakończeniu obsługi klienta, serwer powinien nieprzerwanie oczekiwać na kolejnych klientów, których będzie mógł obsłużyć. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Numer portu TCP, na którym ma działać serwer, pobierz z linii poleceń. Zadbaj o prawidłową obsługę błędów.

**4.4** Zmodyfikuj program z zadania 4.3 w taki sposób, aby serwer działał pod adresem IPv6. Po zakończeniu obsługi klienta, serwer powinien nieprzerwanie oczekiwać na kolejnych klientów, których będzie mógł obsłużyć. Numer portu TCP pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**4.5** Napisz program serwera, który działając pod adresem IPv4 oraz na podanym porcie TCP, będzie losował liczbę i odbierał od klienta wiadomości. W przypadku, gdy w wiadomości klient przysłał do serwera coś innego, niż liczbę, serwer powinien poinformować klienta o błędzie. Po odebraniu liczby od klienta, serwer sprawdza, czy otrzymana liczba jest:

- mniejsza od wylosowanej przez serwer
- równa wylosowanej przez serwer
- większa od wylosowanej przez serwer

A następnie odsyła stosowną informację do klienta. W przypadku, gdy klient odgadnie liczbę, serwer powinien zakończyć działanie. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Numer portu TCP, na którym ma działać serwer, pobierz z linii poleceń. Zadbaj o prawidłową obsługę błędów.

**4.6** Zmodyfikuj program z zadania 4.5 w taki sposób, aby serwer działał pod adresem IPv6. Numer portu TCP pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.



**4.7** Napisz program serwera, który działając pod adresem IPv4 oraz na podanym porcie TCP, będzie odsyłał klientowi jego wiadomość (o maksymalnej długości 20 znaków) (tzw. serwer echa). Uwzględnij sytuacje, gdy:

- wiadomość do wysłania jest za krótka - ma być wówczas uzupełniana do 20 znaków znakami spacji
- wiadomość do wysłania jest za długa - ma być przycięta do 20 znaków (lub wysłana w całości - sprawdź, co się wówczas stanie)

Numer portu TCP pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**4.8** Zmodyfikuj program z zadania 4.7 w taki sposób, aby serwer działał pod adresem IPv6. Numer portu TCP pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**4.9** W implementacji gniazd, funkcje wysyłania i odbierania danych (np. funkcje `recv` i `send` w Pythonie) nie gwarantują wysłania / odbioru wszystkich danych. Rozważmy funkcję `recv`. Przykładowo, 100 bajtów może zostać wysłane jako grupa po 10 bajtów, albo od razu w całości. Oznacza to, iż jeśli używamy gniazd TCP, musimy odbierać dane, dopóki nie mamy pewności, że odebraliśmy odpowiednią ich ilość.

Napisz program serwera, który działając pod adresem IPv4 oraz na podanym porcie TCP, dla podłączającego się klienta, będzie odsyłał mu przesłaną wiadomość (tzw. serwer echa). Prawidłowa komunikacja powinna odbywać się w następujący sposób:

- Serwer odbiera dane od klienta
- Serwer odsyła klientowi odebrane od niego dane (identyczną wiadomość)

**Aby ułatwić sobie zadanie wysyłania i odbierania CAŁEJ wiadomości, możesz każdą z wysłanych / odebranych wiadomości kończyć znakiem nowej linii - `\n`.** Po zakończeniu obsługi klienta, serwer powinien nieprzerwanie oczekiwać na kolejnych klientach, których będzie mógł obsłużyć. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Numer portu TCP, na którym ma działać serwer, pobierz z linii poleceń. Zadbaj o prawidłową obsługę błędów.

**4.10** Zmodyfikuj program z zadania 4.9 w taki sposób, aby serwer działał pod adresem IPv6. Numer portu TCP pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

## Gniazda UDP

**4.11** Napisz program serwera, który działając pod adresem IPv4 oraz na podanym porcie UDP, dla podłączającego się klienta, będzie odsyłał mu przesłaną wiadomość (tzw. serwer echa). Prawidłowa komunikacja powinna odbywać się w następujący sposób:

- Serwer odbiera dane od klienta
- Serwer odsyła klientowi odebrane od niego dane (identyczną wiadomość)

Po zakończeniu obsługi klienta, serwer powinien nieprzerwanie oczekiwać na kolejnych klientach, których będzie mógł obsłużyć. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Numer portu UDP, na którym ma działać serwer, pobierz z linii poleceń. Zadbaj o prawidłową obsługę błędów.

**4.12** Zmodyfikuj program z zadania 4.11 w taki sposób, aby serwer działał pod adresem IPv6. Numer portu UDP pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

**4.13** Napisz program serwera, który działając pod adresem IPv4 oraz na podanym porcie UDP, dla podłączającego się klienta, będzie odbierał od niego liczbę, operator i liczbę, a następnie odsyłał klientowi wynik działania przez niego przesłanego. Komunikacja powinna wyglądać w następujący sposób:

- Klient łączy się do serwera
- Klient przesyła do serwera pierwszą liczbę
- Serwer odsyła klientowi tekst OK
- Klient wysyła znak działania do serwera (dla uproszczenia, dozwolone są jedynie operacje + - \* /)
- Serwer odsyła klientowi tekst OK
- Klient przesyła do serwera drugą liczbę
- Serwer odsyła klientowi wynik działania

Po zakończeniu obsługi klienta, serwer powinien nieprzerwanie oczekiwać na kolejnych klientów, których będzie mógł obsłużyć. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Numer portu UDP, na którym ma działać serwer, pobierz z linii poleceń. Zadbaj o prawidłową obsługę błędów.

- 4.14** Zmodyfikuj program z zadania **4.13** w taki sposób, aby serwer działał pod adresem IPv6. Numer portu UDP pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

## 5 Bezpiecznie gniazda

## 6 Protokoły pocztowe

### Protokół SMTP

- 6.1 Pod podanym adresem i numerem portu działa serwer obsługujący protokół ESMTP/SMTP. Wykorzystując gotowe oprogramowanie klienckie połącz się z serwerem działającym pod podanym adresem i numerem portu, a następnie wyślij wiadomość e-mail korzystając z komend protokołu ESMTP. *(Zadanie nie jest zadaniem programistycznym, wymaga jedynie użycia gotowego klienta protokołu SMTP w celu zapoznania się z podstawowymi poleceniami protokołu.)*
- 6.2 Pod podanym adresem i numerem portu działa serwer obsługujący protokół ESMTP/SMTP. Wykorzystując gotowe oprogramowanie klienckie połącz się z serwerem działającym pod podanym adresem i numerem portu, a następnie wyślij kilka wiadomości e-mail do kilku odbiorców korzystając z komend protokołu ESMTP. *(Zadanie nie jest zadaniem programistycznym, wymaga jedynie użycia gotowego klienta protokołu SMTP w celu zapoznania się z podstawowymi poleceniami protokołu.)*
- 6.3 Pod podanym adresem i numerem portu działa serwer obsługujący protokół ESMTP/SMTP. Wykorzystując gotowe oprogramowanie klienckie połącz się z serwerem działającym pod podanym adresem i numerem portu, a następnie wyślij wiadomość spoofed e-mail (z podmienionym adresem nadawcy) korzystając z komend protokołu ESMTP. *(Zadanie nie jest zadaniem programistycznym, wymaga jedynie użycia gotowego klienta protokołu SMTP w celu zapoznania się z podstawowymi poleceniami protokołu.)*
- 6.4 Pod podanym adresem i numerem portu działa serwer obsługujący protokół ESMTP/SMTP. Wykorzystując gotowe oprogramowanie klienckie połącz się z serwerem działającym pod podanym adresem i numerem portu, a następnie wyślij wiadomość e-mail korzystając z komend protokołu ESMTP. Do wiadomości dodaj załącznik - dowolny plik tekstowy (sprawdź format MIME: Multipart i Content-Type). Możesz wykorzystać `openssl` do przekonwertowania pliku: `cat plik.txt | openssl base64`. *(Zadanie nie jest zadaniem programistycznym, wymaga jedynie użycia gotowego klienta protokołu SMTP w celu zapoznania się z podstawowymi poleceniami protokołu.)*
- 6.5 Pod podanym adresem i numerem portu działa serwer obsługujący protokół ESMTP/SMTP. Wykorzystując gotowe oprogramowanie klienckie połącz się z serwerem działającym pod podanym adresem i numerem portu, a następnie wyślij wiadomość e-mail korzystając z komend protokołu ESMTP. Do wiadomości dodaj załącznik - dowolny obrazek (sprawdź format MIME: Multipart i Content-Type). Możesz wykorzystać `openssl` do przekonwertowania obrazka: `cat obrazek | openssl base64`. *(Zadanie nie jest zadaniem programistycznym, wymaga jedynie użycia gotowego klienta protokołu SMTP w celu zapoznania się z podstawowymi poleceniami protokołu.)*
- 6.6 Napisz program klienta, który połączy się z serwerem obsługującym protokół ESMTP/SMTP, działającym na podanym porcie, a następnie wyśle wiadomość e-mail używając komend protokołu ESMTP. O adres nadawcy, odbiorcy (odbiorców), temat wiadomości i jej treść zapytaj użytkownika. Adres serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 6.7 Napisz program klienta, który połączy się z serwerem obsługującym protokół ESMTP/SMTP, działającym na podanym porcie, a następnie wyśle wiadomość e-mail używając komend protokołu ESMTP. Do wiadomości dodaj załącznik - dowolny plik tekstowy (sprawdź format MIME: Multipart i Content-Type). O adres nadawcy, odbiorcy (odbiorców), temat wiadomości i jej treść zapytaj użytkownika. Adres serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 6.8 Napisz program klienta, który połączy się z serwerem obsługującym protokół ESMTP/SMTP, działającym na podanym porcie, a następnie wyśle wiadomość e-mail używając komend protokołu ESMTP. Do wiadomości dodaj załącznik - dowolny obrazek (sprawdź format MIME: Multipart i Content-Type). O adres nadawcy, odbiorcy (odbiorców), temat wiadomości i jej treść zapytaj użytkownika. Adres serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

- 6.9** Napisz program klienta, który połączy się z serwerem obsługującym protokół ESMTP/SMTP, działającym na podanym porcie, a następnie wyśle wiadomość e-mail używając komend protokołu ESMTP. Do wiadomości dodaj załącznik - dowolny plik dźwiękowy. (sprawdź format MIME: Multipart i Content-Type). O adres nadawcy, odbiorcy (odbiorców), temat wiadomości i jej treść zapytaj użytkownika. Adres serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 6.10** Napisz program klienta, który połączy się z serwerem obsługującym protokół ESMTP/SMTP, działającym na podanym porcie, a następnie wyśle wiadomość e-mail używając komend protokołu ESMTP. Treść wiadomości powinna zostać sformatowana za pomocą tagów HTML, przykładowo: **<b>pogrubienie</b>**, *<i>pochylenie</i>*, <u>podkreślenie</u> i innych wybranych. O adres nadawcy, odbiorcy (odbiorców), temat wiadomości i jej treść zapytaj użytkownika. Adres serwera i numer portu pobierz jako argumenty linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 6.11** Napisz program serwera, który działając pod adresem IPv4 127.0.0.1 oraz na określonym porcie TCP, będzie serwerem poczty, obsługującym protokół SMTP. Nie realizuj faktycznego wysyłania e-maila, tylko zasymuluj jego działanie tak, żeby napisany wcześniej klient SMTP myślał, że wiadomość została wysłana. Pamiętaj o obsłudze przypadku, gdy klient poda nie zaimplementowaną przez serwer komendę. Numer portu pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 6.12** Zmodyfikuj program z zadania **6.11** w taki sposób, aby łączył serwer działał pod adresem IPv6 - ::1. Numer portu pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

## Protokół POP3

## Protokół IMAP

## 7 Protokół HTTP (w wersji 1.1)

**Protokół HTTP (*Hypertext Transfer Protocol*, RFC 2616 oraz od 7230 do 7235)** - protokół warstwy aplikacji, wykorzystujący na niższej warstwie (zazwyczaj) gniazda TCP/IP oraz 2 domyślne porty: port niezabezpieczony 80 i port zabezpieczony: 443.

Podstawowe informacje:

- Protokół HTTP jest protokołem wykorzystywanym do przesyłania plików (ogólnie mówiąc: zasobów) w sieci WWW (World Wide Web), bez względu na to, czy zasobem jest plik HTML, plik graficzny, wynik zapytania, czy cokolwiek innego
- Protokół HTTP, do wersji 1.1, jest protokołem tekstowym, gdzie komendy protokołu, podobnie jak w SMTP, POP3 czy IMAP są komendami tekstowymi, zrozumiałymi dla człowieka
- HTTP to protokół typu zapytanie-odpowiedź. Zapytanie, wysyłane przez klienta, zawiera informację o żądanym zasobie. Odpowiedź, wysyłana przez serwer, zawiera treść zasobu. Jeśli serwer nie jest w stanie zwrócić odpytywanego zasobu, odpowiedź zawiera kod reprezentujący powód, dla którego zasób nie mógł być wysłany (np. zasób nie istnieje)
- Formaty zapytania i odpowiedzi HTTP są do siebie podobne; zarówno zapytanie, jak i odpowiedź HTTP zawierają (linia początkowa i nagłówki powinny się kończyć parą znaków CRLF, czyli `\r\n`):
  - linię początkową
  - 0 lub więcej nagłówków
  - pustą linię (CRLF, czyli `\r\n`)
  - opcjonalne ciało wiadomości

**Przykład:**

```
linia początkowa, inna dla zadanania, inna dla odpowiedzi \r\n
naglowek1: wartosc1 \r\n
naglowek2: wartosc2 \r\n
naglowek3: wartosc3 \r\n
\r\n
ciało wiadomosci, może się składać z 1 lub
wielu linii, lub może być puste
```

- Nagłówki HTTP to wszelkie komendy używane do komunikacji między przeglądarką WWW (klientem) a serwerem. Nagłówki są to właściwości żądania i odpowiedzi przesyłane wraz z samą wiadomością. Służą one przede wszystkim do sterowania zachowaniem serwera oraz przeglądarki przez nadawcę wiadomości.
- Jeśli klient wysyła żądanie do serwera HTTP, żądanie powinno zawsze być zakończone parą znaków CRLF (czyli `\r\n`)
- Serwer odsyłając odpowiedź HTTP nie określa za pomocą żadnych specjalnych znaków końca odsyłanej odpowiedzi. W przypadku, gdy chcemy mieć pewność, że odebraliśmy całą odpowiedź serwera HTTP, musimy parsować odebrane nagłówki (**Content-Length** lub **Transfer-Encoding**), w których może znajdować się informacja o tym, jaki jest rozmiar odpowiedzi serwera, i użyć tej informacji do odebrania całej wiadomości. W przypadku, gdy serwer w odpowiedzi HTTP nie odeśle żadnego z powyższych nagłówków, aby mieć pewność odebrania całej odpowiedzi od serwera, musimy odbierać dane, dopóki serwer nie zakończy / zamknie połączenia. Zgodnie z formatem żądania i odpowiedzi HTTP, nagłówki od ciała oddzielają znaki CRLF CRLF (czyli `\r\n \r\n`).



Żądania HTTP

Ogólny format *żądania* HTTP (pola oddzielone spacjami):

```
Method Request-URI HTTP-Version \r\n
HEADER1: VALUE1 \r\n
HEADER2: VALUE2 \r\n
...
HEADERX: VALUEX \r\n
\r\n
BODY
\r\n
```

gdzie:

- **Method** - to metoda żądania, dozwolone metody HTTP:
  - \* **GET** – pobranie zasobu wskazanego przez **Request-URI**
  - \* **HEAD** – pobiera informacje o zasobie, stosowane do sprawdzania dostępności zasobu
  - \* **PUT** – przyjęcie danych przesyłanych od klienta do serwera, najczęściej aby zaktualizować wartość zasobu,
  - \* **POST** – przyjęcie danych przesyłanych od klienta do serwera (np. wysyłanie zawartości formularzy),
  - \* **DELETE** – żądanie usunięcia zasobu,
  - \* **OPTIONS** – informacje o opcjach i wymaganiach dotyczących zasobu,
  - \* **TRACE** – diagnostyka, analiza kanału komunikacyjnego,
  - \* **CONNECT** – żądanie przeznaczone dla serwerów pośredniczących pełniących funkcje tunelowania,
  - \* **PATCH** – aktualizacja części zasobu (np. jednego pola).
- **Request-URI** - to ścieżka do zasobu na serwerze, która może zawierać dodatkowo parametry HTTP oraz fragment (za znakiem #),
- **HTTP-Version** - wersja protokołu HTTP, np. HTTP/1.0, HTTP/1.1, HTTP/2.0
- **HEADER1**, **HEADER2**, ..., **HEADERX** - nagłówki HTTP, **VALUE1**, **VALUE2**, ..., **VALUEX** - wartości konkretnych nagłówków
- **BODY** - opcjonalne ciało żądania

Odpowiedzi HTTP

Ogólny format **odpowiedzi** HTTP (pola oddzielone spacjami):

```
HTTP-Version Status-Code Reason-Phrase \r\n
HEADER1: VALUE1 \r\n
HEADER2: VALUE2 \r\n
...
HEADERX: VALUEX \r\n
\r\n
BODY
\r\n
```

gdzie:

**HTTP-Version** - wersja protokołu HTTP, np. HTTP/1.0, HTTP/1.1, HTTP/2.0

**Status-Code** - **kod odpowiedzi**, który informuje klienta, w jaki sposób żądanie zostało lub nie zostało obsłużone, kody odpowiedzi to liczby trzycyfrowe, gdzie pierwsza z nich określa grupę odpowiedzi:

- \* 1xx - to kody informacyjne
- \* 2xx - to kody powodzenia
- \* 3xx - to kody przekierowania
- \* 4xx - to kody błędu aplikacji klienta
- \* 5xx - to kody błędu serwera

**Reason-Phrase** - wiadomość powiązana z danym kodem odpowiedzi

**HEADER1, HEADER2, ..., HEADERX** - nagłówki HTTP, **VALUE1, VALUE2, ..., VALUEX** - wartości konkretnych nagłówków

**BODY** - opcjonalne ciało żądania

Dozwolone nagłówki HTTP:

- **General Header Fields** - are a few header fields which have general applicability for both request and response messages, but which do not apply to the entity being transferred.
- **Entity Header Fields** - define metainformation about the entity-body or, if no body is present, about the resource identified by the request.
- **Request Header Fields** - allow the client to pass additional information about the request, and about the client itself, to the server.
- **Response Header Fields** - allow the server to pass additional information about the response which cannot be placed in the Status- Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

Przykłady żądań i odpowiedzi HTTP:

- Żądanie:

```
GET /index.html HTTP/1.1
HOST: 212.182.24.27
```

- Odpowiedź:

```
HTTP/1.1 200 OK
Date: Thu, 13 Apr 2017 14:25:38 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Thu, 13 Apr 2017 13:57:13 GMT
ETag: "2c39-54d0cb3af4405"
Accept-Ranges: bytes
Content-Length: 11321
Vary: Accept-Encoding
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Apache2 Ubuntu Default Page: It works</title>
  </head>
  <body>
    ...
  </body>
</html>
```

- Żądanie:

```
TRACE / HTTP/1.1
HOST: 212.182.24.27
```

- Odpowiedź:

```
HTTP/1.1 405 Method Not Allowed
Date: Thu, 13 Apr 2017 14:31:22 GMT
Server: Apache/2.4.18 (Ubuntu)
Allow:
Content-Length: 302
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>405 Method Not Allowed</title>
</head><body>
<h1>Method Not Allowed</h1>
<p>The requested method TRACE is not allowed for the URL ./</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at 212.182.24.27 Port 80</address>
</body></html>
```

- Żądanie:

```
OPTIONS /index.html HTTP/1.1
HOST: 212.182.24.27
```

- Odpowiedź:

```
HTTP/1.1 200 OK
Date: Thu, 13 Apr 2017 14:52:31 GMT
Server: Apache/2.4.18 (Ubuntu)
Allow: OPTIONS,GET,HEAD,POST
Content-Length: 0
Content-Type: text/html
```

- Żądanie:

```
HEAD /index.html HTTP/1.1
HOST: 212.182.24.27
```

- Odpowiedź:

```
HTTP/1.1 200 OK
Date: Thu, 13 Apr 2017 14:53:06 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Thu, 13 Apr 2017 13:57:13 GMT
ETag: "2c39-54d0cb3af4405"
Accept-Ranges: bytes
Content-Length: 11321
Vary: Accept-Encoding
Content-Type: text/html
```

- 7.1** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex1:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `7001` działa serwer obsługujący protokół HTTP w wersji 1.1. Pod odnośnikiem `/html` udostępnia prostą stronę HTML. Napisz program klienta, który połączy się z serwerem, a następnie pobierze treść strony i zapisze ją na dysku jako plik z rozszerzeniem `*.html`. Spreparuj żądanie HTTP tak, aby serwer myślał, że żądanie przyszło od przeglądarki `Safari 7.0.3`. Jakich nagłówków HTTP należy użyć? Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.2** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex2:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `7002` działa serwer obsługujący protokół HTTP w wersji 1.1. Pod odnośnikiem `/image/png` udostępnia obrazek. Napisz program klienta, który połączy się z serwerem, a następnie pobierze obrazek i zapisze go na dysku. Jakich nagłówków HTTP należy użyć? Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.3** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex3:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `7003` działa serwer obsługujący protokół HTTP w wersji 1.1. Pod odnośnikiem `/image.jpg` udostępnia obrazek. Napisz program klienta, który połączy się z serwerem, a następnie pobierze z serwera obrazek w 3 częściach i po odebraniu wszystkich części złoży go w całość i zapisze na dysku. Jakich nagłówków HTTP należy użyć? Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.4** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex4:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `7004` działa serwer obsługujący protokół HTTP w wersji 1.1. Pod odnośnikiem `/post` udostępnia formularz z polami do wypełnienia. Napisz program klienta, który połączy się z serwerem, a następnie uzupełni formularz danymi pobranymi od użytkownika, a następnie prześle go do serwera i odbierze odpowiedź.

Aby sprawdzić, jak wygląda żądanie HTTP potrzebne do wypełnienia i wysłania formularza:

- jakie *nagłówki HTTP* są wykorzystywane,
- jak wygląda ciało zapytania,

podśłuchaj komunikację z serwerem za pomocą Wiresharka, tj. uruchom przeglądarkę oraz Wiresharka; uzupełnij i zatwierdź formularz ręcznie za pomocą przeglądarki, a następnie sprawdź pakiety podsłuchane podczas komunikacji z serwerem działającym pod adresem `http://127.0.0.1:7004`. Możesz użyć filtrów Wiresharka: `http.request` oraz `http.response` (`http.request || http.response`). Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

- 7.5 Slowloris, czyli Slow HTTP Headers DoS** Atak o nazwie Slowloris, dzięki wykorzystaniu pewnych koncepcji protokołu HTTP oraz sposobu obsługi żądań serwerów WWW, potrafi całkowicie je sparaliżować w przeciągu kilku sekund. Atak polega na utworzeniu dużej liczby gniazd, a następnie dosyłania w powolny sposób danych częściowych żądań HTTP, co w końcu skutkuje wyczerpaniem puli wolnych wątków obsługujących żądania HTTP.

W klasycznym żądaniu, np. wykorzystującym metodę HTTP `GET`, do serwera wysyłana jest linia żądania, nagłówki oraz pusta linia `CRLF` oznaczająca koniec nagłówków. Atak Slowloris polega na wysyłaniu dużej liczby dodatkowych nagłówków, przykładowo `X-a: b`, które będą sukcesywnie przychodzić do atakowanego serwera dopiero po pewnym czasie. Podsumowując, atak działa następująco:

- (a) Budowane są gniazda TCP (im więcej, tym lepiej, domyślnie 1000)  
`sock = socket(AF_INET, SOCK_STREAM)`
- (b) Następuje podłączenie do serwera i wysyłanie podstawowych nagłówków  
`sock.connect(server), sock.send('...')`,
- (c) Wysyłany jest nagłówek X-a: b \r\n  
`sock.send('...')`
- (d) Oczekujemy pewien czas (domyślnie 100 sekund)  
`time.sleep(100)`
- (e) Wysyłamy ponownie nagłówek X-a: b \r\n  
`sock.send('...')`
- (f) Powtarzamy do skutku kroki 4. i 5. dla każdego połączenia, ewentualnie dobudowujemy gniazda do zamkniętych połączeń

Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex5:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7005 działa serwer obsługujący protokół HTTP w wersji 1.1. Znając założenia ataku Slowloris, napisz program klienta - atakującego, który wykona atak Slowloris na serwer WWW. Jakich nagłówków HTTP należy użyć? Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.

- 7.6** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex6:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7006 działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer udostępnia różne wersje językowe swojej strony głównej. Wykorzystując nagłówki protokołu HTTP, napisz program klienta, który połączy się z serwerem, a następnie pobierze niemiecką wersję strony głównej. Jakich nagłówków HTTP należy użyć? Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.7** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex3:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7003 działa serwer obsługujący protokół HTTP w wersji 1.1. Pod odnośnikiem `/image.jpg` udostępnia obrazek. Napisz program klienta, który połączy się z serwerem, a następnie pobierze z serwera obrazek w 3 częściach i po odebraniu wszystkich części złoży go w całość i zapisze na dysku. **Następnie zmodyfikuj program w taki sposób, aby pobierał z serwera obrazek tylko wtedy, gdy nie zmienił się on od ostatniego pobrania.** Jakich nagłówków HTTP należy użyć? Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.8** Napisz program serwera, który działając pod podanym adresem IPv4 oraz na określonym porcie TCP, będzie serwerem HTTP. Obsłuż wybrane nagłówki i co najmniej jeden kod błędu (np. 404). Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.9** Zmodyfikuj program z zadania 7.8 w taki sposób, aby serwer wykorzystywał adres IPv6. Numer portu pobierz jako argument linii poleceń. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.10** Poniżej znajduje się request protokołu HTTP/1.1 w zapisie szesnastkowym. Wiedząc, że znak \r w zapisie szesnastkowym to 0x0d, natomiast \n to 0x0a oraz wiedząc, że w zapisie szesnastkowym jedna cyfra reprezentuje 4 bity, napisz program, w którym sprawdzisz, jakie nagłówki zostały wysłane do serwera.

```

47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a
20 48 6f 73 74 3a 20 77 77 77 2e 75 6d 63 73 2e
70 6c 20 0d 0a 20 55 73 65 72 2d 41 67 65 6e 74
3a 20 63 75 72 6c 2f 37 2e 38 31 2e 30 20 0d 0a
20 41 63 63 65 70 74 3a 20 2a 2f 2a 20 0d 0a 0d
0a

```

- 7.11** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex11:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7011 działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer w odpowiedzi na żądanie typu GET zwraca parametry żądania. Napisz program klienta, który połączy się z serwerem, a następnie wyśle do serwera żądanie GET z parametrem `key`, który dostaje wartość `value`, odbierze i wyświetli odpowiedź. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.12** Spróbuj rozwiązać zadanie **7.11** za pomocą narzędzia `curl`.
- 7.13** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex13:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7013 działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer w odpowiedzi na odebrany request, odsyła do klienta przesłane przez niego ciasteczko (`cookie header`). W przypadku, gdy klient nie prześle ciasteczka, serwer zwraca odpowiedź z kodem 404. Napisz program klienta, który połączy się z serwerem, a następnie wyśle do serwera ciasteczko z kluczem `key` oraz wartością `value`, a następnie odbierze i wyświetli odpowiedź. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.14** Spróbuj rozwiązać zadanie **7.13** za pomocą narzędzia `curl`.
- 7.15** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex11:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7011 działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer w odpowiedzi na żądanie typu GET zwraca parametry żądania. Napisz program klienta, który połączy się z serwerem, a następnie wyśle do serwera żądanie GET z parametrem `key`, który dostaje wartość `value`, odbierze i wyświetli odpowiedź. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów. **Uwaga: wyślij do serwera 2 parametry key z wartością value.**
- 7.16** Spróbuj rozwiązać zadanie **7.15** za pomocą narzędzia `curl`.
- 7.17** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex11:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7011 działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer w odpowiedzi na żądanie typu GET zwraca parametry żądania. Napisz program klienta, który połączy się z serwerem, a następnie wyśle do serwera żądanie GET z parametrem `key`, który dostaje wartość `value&`, odbierze i wyświetli odpowiedź. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów. **Uwaga: zwróć szczególną uwagę, czy serwer zwróci odpowiednią wartość parametru, czyli `value&` zamiast `value`.**
- 7.18** Spróbuj rozwiązać zadanie **7.17** za pomocą narzędzia `curl`.
- 7.19** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex19:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7019 działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer w odpowiedzi na żądanie typu GET zwraca parametry żądania. Napisz program klienta, który połączy się z serwerem, a następnie wyśle do serwera żądanie GET z parametrem `?key`, który dostaje wartość `value`, odbierze i wyświetli odpowiedź. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów. **Uwaga: zwróć szczególną uwagę, czy serwer zwróci odpowiedni parametr, czyli `?key`.**
- 7.20** Spróbuj rozwiązać zadanie **7.19** za pomocą narzędzia `curl`.

- 7.21** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex11:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `7011` działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer w odpowiedzi na żądanie typu `GET` zwraca parametry żądania. Napisz program klienta, który połączy się z serwerem, a następnie wyśle do serwera żądanie `GET` z parametrem `key`, który dostaje wartość `imie nazwisko`, odbierze i wyświetli odpowiedź. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów. **Uwaga: zwróć szczególną uwagę, czy serwer zwróci odpowiednią wartość parametru key, czyli imie nazwisko.**
- 7.22** Spróbuj rozwiązać zadanie **7.21** za pomocą narzędzia `curl`.
- 7.23** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex11:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `7011` działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer w odpowiedzi na żądanie typu `GET` zwraca parametry żądania. Napisz program klienta, który połączy się z serwerem, a następnie wyśle do serwera żądanie `GET` z parametrem `key`, który dostaje wartość `value#`, odbierze i wyświetli odpowiedź. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów. **Uwaga: zwróć szczególną uwagę, czy serwer zwróci odpowiednią wartość parametru key, czyli value#.**
- 7.24** Spróbuj rozwiązać zadanie **7.23** za pomocą narzędzia `curl`.
- 7.25** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex25:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `7025` działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer w odpowiedzi na żądanie typu `GET` zwraca parametry żądania. Napisz program klienta, który połączy się z serwerem, a następnie wyśle do serwera żądanie `GET` z parametrem `key`, który jest tablicą, gdzie pierwszy element tablicy to `value1`, a drugi element tablicy to `value2`. Następnie klient odbierze i wyświetli odpowiedź. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. Zadbaj o prawidłową obsługę błędów.
- 7.26** Spróbuj rozwiązać zadanie **7.25** bez użycia znaków `[` oraz `]`.
- 7.27** Spróbuj rozwiązać zadanie **7.25** za pomocą narzędzia `curl`.

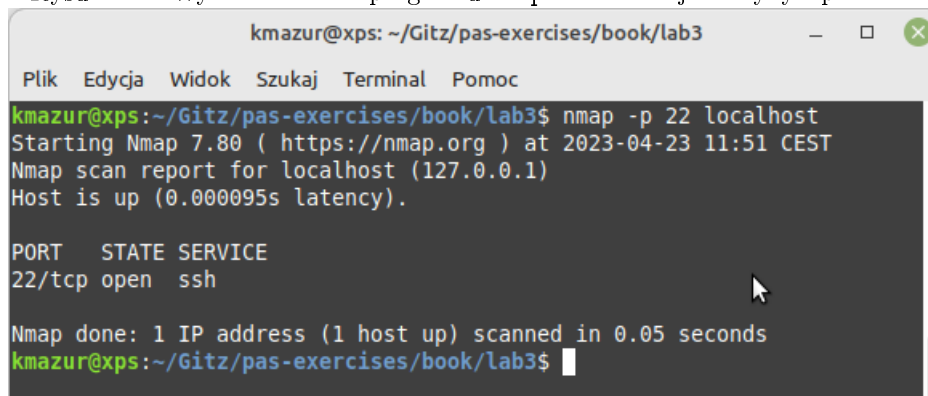


## 8 Odpowiedzi

### Gniazda klienckie TCP

- 3.1** Aby przetestować poprawność swojego rozwiązania, możesz przeskanować swój własny komputer, żeby sprawdzić, czy skanowanie da taki sam wynik, jak Twój program. Do tego celu możesz wykorzystać np. narzędzie `nmap`. Zainstaluj `nmap`: `sudo apt install nmap`, a następnie wydaj polecenie: `nmap -p 22 localhost` (lub `nmap -p 22 127.0.0.1`) aby sprawdzić, czy port 22 jest otwarty na Twoim komputerze. Porównaj wynik z działaniem Twojego programu dla tego samego portu.

Rysunek 1: Wynik działania programu `nmap` dla lokalnej maszyny i portu 22



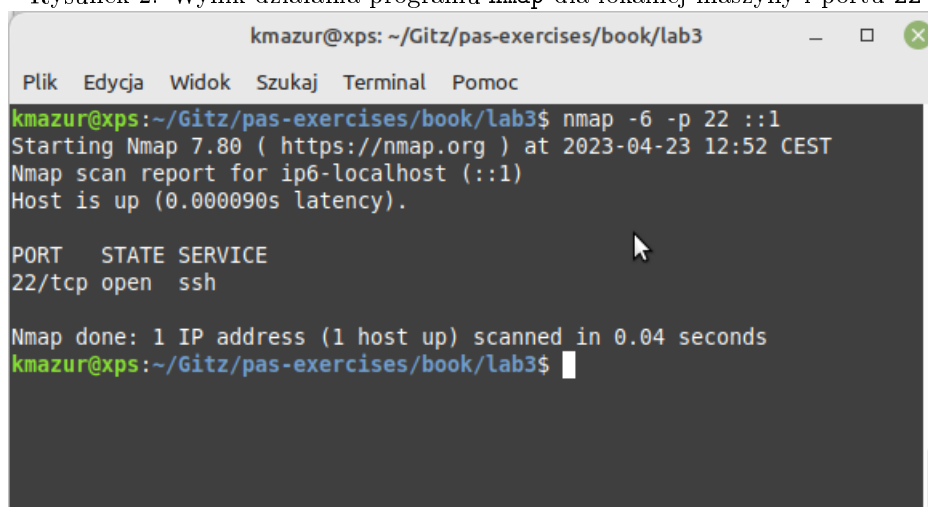
```
kmazur@xps: ~/Gitz/pas-exercises/book/lab3
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
kmazur@xps:~/Gitz/pas-exercises/book/lab3$ nmap -p 22 localhost
Starting Nmap 7.80 ( https://nmap.org ) at 2023-04-23 11:51 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000095s latency).

PORT      STATE SERVICE
22/tcp    open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.05 seconds
kmazur@xps:~/Gitz/pas-exercises/book/lab3$
```

- 3.2** Możesz przetestować program używając swojego lokalnego adresu IPv6: `::1` lub `ip6-localhost` lub `0000:0000:0000:0000:0000:0000:0000:0001`.

Rysunek 2: Wynik działania programu `nmap` dla lokalnej maszyny i portu 22



```
kmazur@xps: ~/Gitz/pas-exercises/book/lab3
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
kmazur@xps:~/Gitz/pas-exercises/book/lab3$ nmap -6 -p 22 ::1
Starting Nmap 7.80 ( https://nmap.org ) at 2023-04-23 12:52 CEST
Nmap scan report for ip6-localhost (::1)
Host is up (0.000090s latency).

PORT      STATE SERVICE
22/tcp    open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
kmazur@xps:~/Gitz/pas-exercises/book/lab3$
```

- 3.3** Aby przetestować poprawność swojego rozwiązania, możesz przeskanować swój własny komputer, żeby sprawdzić, czy skanowanie da taki sam wynik, jak Twój program. Do tego celu możesz wykorzystać np. narzędzie `nmap`. Zainstaluj `nmap`: `sudo apt install nmap`, a następnie wydaj polecenie: `nmap -p1-65535 localhost` (lub `nmap -p1-65535 127.0.0.1`) aby sprawdzić, jakie porty są otwarte na Twoim komputerze. Porównaj wynik z działaniem Twojego programu dla tego samego portu.

Rysunek 3: Wynik działania programu `nmap` dla lokalnej maszyny i wszystkich jej portów

```

kmazur@xps: ~/Gitz/pas-exercises/book/lab3
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
kmazur@xps:~/Gitz/pas-exercises/book/lab3$ nmap -p1-65535 localhost
Starting Nmap 7.80 ( https://nmap.org ) at 2023-04-23 12:36 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00012s latency).
Not shown: 65529 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
443/tcp   open  https
631/tcp   open  ipp
3306/tcp  open  mysql
33060/tcp open  mysqlx

Nmap done: 1 IP address (1 host up) scanned in 2.68 seconds
kmazur@xps:~/Gitz/pas-exercises/book/lab3$

```

- 3.4** Aby przetestować poprawność swojego rozwiązania, możesz przeskanować swój własny komputer, żeby sprawdzić, czy skanowanie da taki sam wynik, jak Twój program. Do tego celu możesz wykorzystać np. narzędzie `nmap`. Zainstaluj `nmap`: `sudo apt install nmap`, a następnie wydaj polecenie: `nmap -6 -p1-65535 ip6-localhost` (lub `nmap -6 -p1-65535 ::1`) aby sprawdzić, jakie porty są otwarte na Twoim komputerze. Porównaj wynik z działaniem Twojego programu dla tego samego portu.

Rysunek 4: Wynik działania programu `nmap` dla lokalnej maszyny i wszystkich jej portów

```

kasiula@dell:~/Gitz/pas-exercises/book$ nmap -6 -p1-65535 ::1
Starting Nmap 7.80 ( https://nmap.org ) at 2023-05-19 20:53 CEST
Nmap scan report for ip6-localhost (::1)
Host is up (0.000087s latency).
Not shown: 65527 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
111/tcp   open  rpcbind
631/tcp   open  ipp
2049/tcp  open  nfs
35855/tcp open  unknown
40071/tcp open  unknown
40143/tcp open  unknown
51369/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 2.20 seconds
kasiula@dell:~/Gitz/pas-exercises/book$

```

- 3.5** Aby przetestować poprawność swojego rozwiązania, możesz skorzystać z gotowego serwera, do którego może połączyć się Twój klient, aby pobrać aktualną datę i czas. Aby uruchomić serwer, zainstaluj Dockera, a następnie w konsoli wydaj polecenia:

- Pobierz obraz serwera:  
`docker pull mazurkatarzyna/pas-book-p1-ch3-ex5-server:latest`
- Uruchom serwer za pomocą Dockera:  
`docker run -dp 3005:3005 mazurkatarzyna/pas-book-p1-ch3-ex5-server`

Tak uruchomiony serwer działa na Twoim komputerze, jego adres IPv4 to `127.0.0.1` (`localhost`), numer portu to `3005`.

**3.6** Aby przetestować poprawność swojego rozwiązania, możesz skorzystać z gotowego serwera, do którego może połączyć się Twój klient, aby pobrać aktualną datę i czas. Aby uruchomić serwer, zainstaluj Dockera, a następnie w konsoli wydaj polecenia:

- Pobierz obraz serwera:  
`docker pull mazurkatarzyna/pas-book-p1-ch3-ex6-server:latest`
- Uruchom serwer za pomocą Dockera:  
`docker run -dp 3006:3006 mazurkatarzyna/pas-book-p1-ch3-ex6-server`

Tak uruchomiony serwer działa na Twoim komputerze, jego adres IPv6 to `:::1 (localhost)`, numer portu to 3006.

**3.7** Aby przetestować poprawność swojego rozwiązania, możesz skorzystać z gotowego serwera, do którego może połączyć się Twój klient, aby wysłać wiadomość. Aby uruchomić serwer, zainstaluj Dockera, a następnie w konsoli wydaj polecenia:

- Pobierz obraz serwera:  
`docker pull mazurkatarzyna/pas-book-p1-ch3-ex7-server:latest`
- Uruchom serwer za pomocą Dockera:  
`docker run -dp 3007:3007 mazurkatarzyna/pas-book-p1-ch3-ex7-server`

Tak uruchomiony serwer działa na Twoim komputerze, jego adres IPv4 to `127.0.0.1 (localhost)`, numer portu to 3007.

**3.8** Aby przetestować poprawność swojego rozwiązania, możesz skorzystać z gotowego serwera, do którego może połączyć się Twój klient, aby wysłać wiadomość. Aby uruchomić serwer, zainstaluj Dockera, a następnie w konsoli wydaj polecenia:

- Pobierz obraz serwera:  
`docker pull mazurkatarzyna/pas-book-p1-ch3-ex8-server:latest`
- Uruchom serwer za pomocą Dockera:  
`docker run -dp 3008:3008 mazurkatarzyna/pas-book-p1-ch3-ex8-server`

Tak uruchomiony serwer działa na Twoim komputerze, jego adres IPv6 to `:::1 (localhost)`, numer portu to 3008.

**3.9** Aby przetestować poprawność swojego rozwiązania, możesz skorzystać z gotowego serwera, do którego może połączyć się Twój klient, aby wysłać wiadomość. Aby uruchomić serwer, zainstaluj Dockera, a następnie w konsoli wydaj polecenia:

- Pobierz obraz serwera:  
`docker pull mazurkatarzyna/pas-book-p1-ch3-ex9-server:latest`
- Uruchom serwer za pomocą Dockera:  
`docker run -dp 3009:3009 mazurkatarzyna/pas-book-p1-ch3-ex9-server`

Tak uruchomiony serwer działa na Twoim komputerze, jego adres IPv4 to `127.0.0.1 (localhost)`, numer portu to 3009.

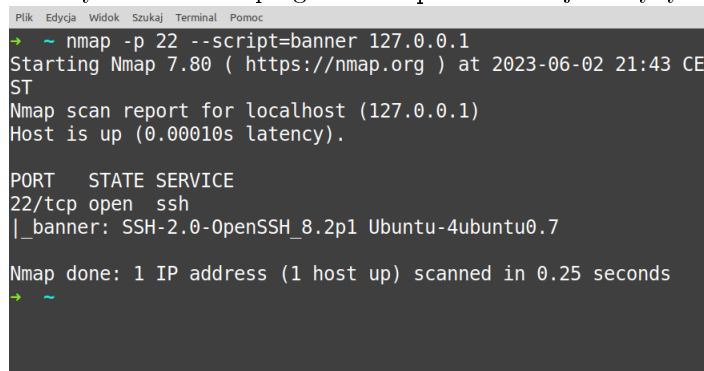
**3.10** Aby przetestować poprawność swojego rozwiązania, możesz skorzystać z gotowego serwera, do którego może połączyć się Twój klient, aby wysłać wiadomość. Aby uruchomić serwer, zainstaluj Dockera, a następnie w konsoli wydaj polecenia:

- Pobierz obraz serwera:  
`docker pull mazurkatarzyna/pas-book-p1-ch3-ex10-server:latest`
- Uruchom serwer za pomocą Dockera:  
`docker run -dp 3010:3010 mazurkatarzyna/pas-book-p1-ch3-ex10-server`

Tak uruchomiony serwer działa na Twoim komputerze, jego adres IPv6 to `:::1 (localhost)`, numer portu to 3010.

- 3.11** Aby przetestować poprawność swojego rozwiązania, możesz przeskanować swój własny komputer, żeby sprawdzić, czy skanowanie da taki sam wynik, jak Twój program. Do tego celu możesz wykorzystać np. narzędzie `nmap` z parametrem `--script=banner`. Zainstaluj `nmap`: `sudo apt install nmap`, a następnie wydaj polecenie: `nmap -p 22 --script=banner localhost` (lub `nmap -p 22 --script=banner 127.0.0.1`) aby sprawdzić, czy port 22 jest otwarty na Twoim komputerze, i jaka usługa jest uruchomiona na tym porcie. Porównaj wynik z działaniem Twojego programu dla tego samego portu.

Rysunek 5: Wynik działania programu `nmap` dla lokalnej maszyny i portu 22



```

Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
➔ ~ nmap -p 22 --script=banner 127.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-02 21:43 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00010s latency).

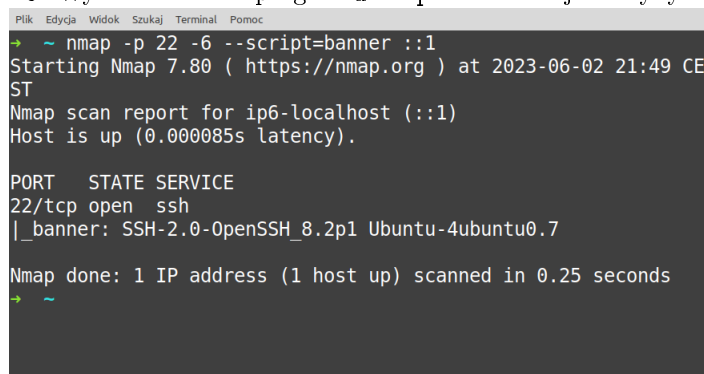
PORT      STATE SERVICE
22/tcp    open  ssh
|_ banner: SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.7

Nmap done: 1 IP address (1 host up) scanned in 0.25 seconds
➔ ~

```

- 3.12** Aby przetestować poprawność swojego rozwiązania, możesz przeskanować swój własny komputer, żeby sprawdzić, czy skanowanie da taki sam wynik, jak Twój program. Do tego celu możesz wykorzystać np. narzędzie `nmap` z parametrem `--script=banner`. Zainstaluj `nmap`: `sudo apt install nmap`, a następnie wydaj polecenie: `nmap -p 22 -6 --script=banner ip6-localhost` (lub `nmap -p 22 -6 --script=banner ::1`) aby sprawdzić, czy port 22 jest otwarty na Twoim komputerze, i jaka usługa jest uruchomiona na tym porcie. Porównaj wynik z działaniem Twojego programu dla tego samego portu.

Rysunek 6: Wynik działania programu `nmap` dla lokalnej maszyny i portu 22



```

Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
➔ ~ nmap -p 22 -6 --script=banner ::1
Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-02 21:49 CEST
Nmap scan report for ip6-localhost (::1)
Host is up (0.000085s latency).

PORT      STATE SERVICE
22/tcp    open  ssh
|_ banner: SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.7

Nmap done: 1 IP address (1 host up) scanned in 0.25 seconds
➔ ~

```

- 3.13** Aby przetestować poprawność swojego rozwiązania, możesz skorzystać z gotowego serwera, do którego może połączyć się Twój klient, aby wysłać wiadomość. Aby uruchomić serwer, zainstaluj Dockera, a następnie w konsoli wydaj polecenia:

- Pobierz obraz serwera:  
`docker pull mazurkatarzyna/pas-book-p1-ch3-ex13-server:latest`
- Uruchom serwer za pomocą Dockera:  
`docker run -dp 3013:3013 mazurkatarzyna/pas-book-p1-ch3-ex13-server`

Tak uruchomiony serwer działa na Twoim komputerze, jego adres IPv4 to `127.0.0.1` (`localhost`), numer portu to `3013`.





```

0030 - c0 9d 2f 95 e4 b3 78 24-98 b8 cd b4 83 50 08 43  ../...x$.....P.C
0040 - 1b e8 2b eb cd ee 3d 20-ef d6 4e 89 9d a6 21 92  ..+...= ..N...!.
0050 - a7 0e 9b 04 e1 1e 9f 36-3f 6d 7b a5 a9 37 7d af  ....6?m{...7}.
0060 - b4 3a 52 dc 68 7a 35 3b-89 74 89 ea ae 34 ba 46  .:R.hz5;t...4.F
0070 - fb e6 c6 da c3 f9 c1 89-f4 1e 89 28 50 32 12 74  .....(P2.t
0080 - 07 55 d0 7d 70 2f 1d e0-02 ca 6a e7 41 25 8f 53  .U.)p/...j.A%.S
0090 - 5e 67 1d 25 f5 9f ab f9-e5 ce 0d 4d 05 78 46 4f  ^g.%.....M.xFO
00a0 - 11 d4 bb 3b 84 d2 26 71-28 3d 0b 13 a9 aa 85 95  ...;...&q(=.....
00b0 - 8d 77 c9 d0 be e5 d3 38-51 b1 1b f2 8b 9f 7e 8c  .w.....8Q.....~.
00c0 - bb e9 5a f4 f9 71 40 3d-a1 1c 5b fe 35 c3 74 bf  ..Z...q@=..[.5.t.

Start Time: 1685969504
Timeout   : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
Max Early Data: 0
---
read R BLOCK
220 ESMTP INTERIA.PL
ehlo student
250-poczta.interia.pl
250-PIPELINING
250-SIZE 157286400
250-AUTH PLAIN LOGIN PLAIN LOGIN PLAIN LOGIN
250-AUTH=PLAIN LOGIN PLAIN LOGIN PLAIN LOGIN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-SMTPUTF8
250 CHUNKING
auth login
334 VXNlcm5hbWU6
a29jaGFtLnVtY3NAaW50ZXJpYS5wbA==
334 UGFzc3dvcnQ6
a29jaGFtLnVtY3MyMDIz
235 2.7.0 Authentication successful
mail from: <kocham.umcs@interia.pl>
250 2.1.0 Ok
rcpt to: <pas.umcs@poczta.fm>
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
to: <pas.umcs@poczta.fm>
from: <kocham.umcs@interia.pl>
subject: test email

Hello!

.
250 OK. ID: 1054c830c3886b54
quit
221 2.0.0 Bye
closed

```

- 6.2** Możesz skorzystać z jednego z serwerów SMTP zaproponowanego w rozwiązaniu do zadania **6.1**. Jaka komenda protokołu SMTP pozwala na określenie odbiorcy wiadomości?
- 6.3** Możesz skorzystać z jednego z serwerów SMTP zaproponowanego w rozwiązaniu do zadania **6.1**. Jaka komenda protokołu SMTP pozwala na określenie nadawcy wiadomości?
- 6.4** Możesz skorzystać z jednego z serwerów SMTP zaproponowanego w rozwiązaniu do zadania **6.1**. Pamiętaj, że każda nowa linia ma znaczenie. Jeśli nie zachowasz odpowiedniej składni, serwer może "nie zrozumieć" wiadomości. Poniżej przykładowe rozwiązanie.

```

openssl s_client -crlf -connect poczta.interia.pl:465
CONNECTED(00000003)
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, CN = DigiCert TLS Hybrid ECC SHA384 2020 CA1
verify return:1
depth=0 C = PL, L = Krakow, O = Grupa INTERIA.PL sp. z o.o. sp. k., CN = *.interia.pl
verify return:1
---
Certificate chain
0 s:C = PL, L = Krakow, O = Grupa INTERIA.PL sp. z o.o. sp. k., CN = *.interia.pl
i:C = US, O = DigiCert Inc, CN = DigiCert TLS Hybrid ECC SHA384 2020 CA1
1 s:C = US, O = DigiCert Inc, CN = DigiCert TLS Hybrid ECC SHA384 2020 CA1
i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIFYzCCBoigAwIBAgIQD/Ztrk8kirkvvn+QyffFMbjAKBgqhkJOPQDzBWMQsw
CQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMTAwLgYDVQQDEyEaWdp
Q2VydCBUTFMgSHlicmlkIEVDQyBTSEEEzODQgMjAyMjAyMjAyMjAyMjAyMjAyMjAy
MDAwHhcNMjAyMjAyMjAyMjAyMjAyMjAyMjAyMjAyMjAyMjAyMjAyMjAyMjAyMjAy
a293MSswKQYDVQQKEyJHcnV5SBjBTIRFVkb1BLb1B1IHVwL1B1IG8ub3Y4c3AuIGsu

```

---

Programowanie aplikacji sieciowych, część 1 | Katarzyna Mazur
40



```

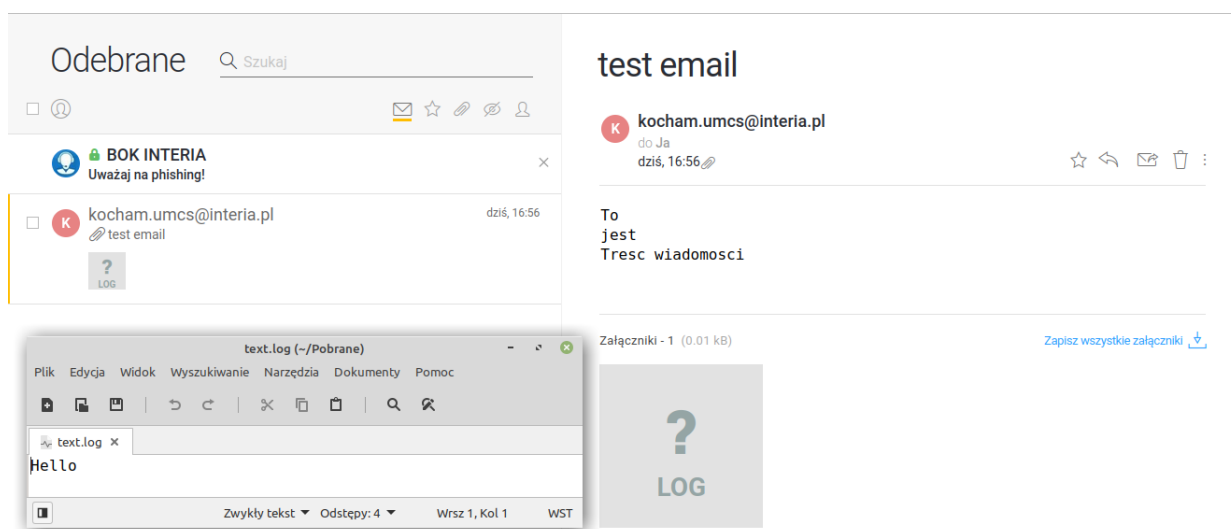
250 CHUNKING
auth login
334 VXNlcm5hbWU6
a29jaGFtLnVtY3NAaW50ZXJpYS5wbA==
334 UGFzc3dvcmQ6
a29jaGFtLnVtY3MyMDIz
235 2.7.0 Authentication successful
mail from: <kocham.umcs@interia.pl>
250 2.1.0 Ok
rcpt to: <pas.umcs@poczta.fm>
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
to: <pas.umcs@poczta.fm>
from: <kocham.umcs@interia.pl>
subject: test email
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=sep

--sep

To
jest
Tresc wiadomosci
--sep
Content-Type: text/x-log; name=text.log
Content-Disposition: attachment; filename=text.log
Content-Transfer-Encoding: base64
SGVsbG8K
--sep--
.
250 OK. ID: bb63e06c9c318562
quit
221 2.0.0 Bye
closed

```

Po sprawdzeniu swojej skrzynki pocztowej, powinieneś zobaczyć wysłaną wiadomość wraz z załącznikiem:



- 6.5** Możesz skorzystać z jednego z serwerów SMTP zaproponowanego w rozwiązaniu do zadania **6.1**. Pamiętaj, że każda nowa linia ma znaczenie. Jeśli nie zachowasz odpowiedniej składni, serwer może "nie zrozumieć" wiadomości. Poniżej przykładowe rozwiązanie.

```

telnet interia.pl 587
Trying 217.74.65.23...
Connected to interia.pl.
Escape character is '^]'.
220 ESMTP INTERIA.PL
HELO student
AUTH LOGIN
cGFzMjAxN0BpbmRlcmhLnBs
UDRTSW5mMjAxNw==
MAIL FROM: <pas2017@interia.pl>
RCPT TO: <pas2017@interia.pl>
RCPT TO: <kasiula.mazur@gmail.com>
DATA

From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>

```



**6.6** Możesz skorzystać z jednego z serwerów SMTP zaproponowanego w rozwiązaniu do zadania **6.1**. Poniżej przykład nawiązania połączenia z serwerem (połączenie szyfrowane), i odebrania od serwera pierwszej wiadomości po połączeniu:

```
#!/bin/usr/env python3
import socket, ssl

def recv_all_until(secure_sock, crlf):
    data = ""
    while not data.endswith(crlf):
        data = data + secure_sock.read(1).decode()
    return data

if __name__ == '__main__':
    HOST = 'poczta.interia.pl'
    PORT = 465

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((HOST, PORT))

    context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
    secure_sock = context.wrap_socket(sock)

    print(recv_all_until(secure_sock, '\r\n'))

    secure_sock.close()
    sock.close()
```

**6.7** Możesz skorzystać z jednego z serwerów SMTP zaproponowanego w rozwiązaniu do zadania **6.1**.

## Protokół HTTP

**7.1** Serwer możesz uruchomić za pomocą poniższych poleceń:

```
docker pull mazurkatarzyna/pas-book-p1-ch7-ex1:latest
docker run -dp 7001:80 mazurkatarzyna/pas-book-p1-ch7-ex1
```

Działanie serwera możesz przetestować za pomocą przeglądarki. W swojej przeglądarce otwórz stronę: <http://127.0.0.1:7001/html>

**7.2** Serwer możesz uruchomić za pomocą poniższych poleceń:

```
docker pull mazurkatarzyna/pas-book-p1-ch7-ex2:latest
docker run -dp 7002:80 mazurkatarzyna/pas-book-p1-ch7-ex2
```

Działanie serwera możesz przetestować za pomocą przeglądarki. W swojej przeglądarce otwórz stronę: <http://127.0.0.1:7002/image/png>

«««< HEAD =====

**7.3** Serwer możesz uruchomić za pomocą poniższych poleceń:

```
docker pull mazurkatarzyna/pas-book-p1-ch7-ex3:latest
docker run -dp 7003:80 mazurkatarzyna/pas-book-p1-ch7-ex3
```

Działanie serwera możesz przetestować za pomocą przeglądarki. W swojej przeglądarce otwórz stronę: <http://127.0.0.1:7003/image.jpg>

**7.4** Serwer możesz uruchomić za pomocą poniższych poleceń:

```
docker pull mazurkatarzyna/pas-book-p1-ch7-ex4:latest
docker run -dp 7004:80 mazurkatarzyna/pas-book-p1-ch7-ex4
```

Działanie serwera możesz przetestować za pomocą przeglądarki. W swojej przeglądarce otwórz stronę: <http://127.0.0.1:7004/post>

**7.5** Serwer możesz uruchomić za pomocą poniższych poleceń:

```
docker pull mazurkatarzyna/pas-book-p1-ch7-ex5:latest
docker run -dp 7005:80 mazurkatarzyna/pas-book-p1-ch7-ex5
```

Działanie serwera możesz przetestować za pomocą przeglądarki. W swojej przeglądarce otwórz stronę: <http://127.0.0.1:7005>. Działający serwer powinien wyświetlać obrazek. W przypadku ataku DDoS, serwer będzie miał trudności w załadowaniu strony. Aby przetestować, czy Twój atak DDoS typu Slowloris jest skuteczny, możesz wykorzystać gotowe narzędzia, np. <https://github.com/gkbrk/slowloris> i zaatakować serwer.

**7.6**