

# Programowanie aplikacji sieciowych

Zbiór zadań, część pierwsza

Katarzyna Mazur

7 lipca 2023

## Spis treści

<b>1</b>	<b>Zadania wprowadzające</b>	<b>6</b>
<b>2</b>	<b>Analiza pakietów sieciowych</b>	<b>7</b>
<b>3</b>	<b>Gniazda klienckie</b>	<b>9</b>
<b>4</b>	<b>Gniazda serwerowe</b>	<b>12</b>
<b>5</b>	<b>Bezpiecznie gniazda</b>	<b>13</b>
<b>6</b>	<b>Protokoły pocztowe</b>	<b>14</b>
<b>7</b>	<b>Protokół HTTP</b>	<b>18</b>
<b>8</b>	<b>Odpowiedzi</b>	<b>20</b>

## 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()
```

## 1 Zadania wprowadzające

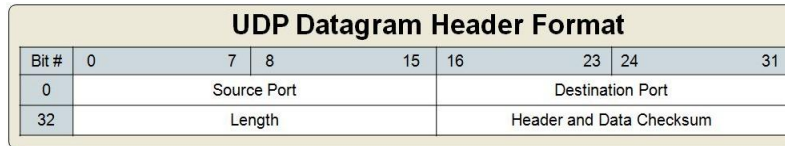
- 1.1 Napisz program, w którym pobierzesz od użytkownika nazwę pliku tekstowego w formacie `*.txt`, a następnie skopiujesz go do pliku pod nazwą `lab1zad1.txt`. Zadbaj o prawidłową obsługę błędów.
- 1.2 Napisz program, w którym pobierzesz od użytkownika nazwę pliku graficznego w formacie `*.png`, a następnie 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 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** Dostępne dla gniazd funkcje `recv` i `send` 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. Podczas pisania kodu związanego z operacjami sieciowymi, nie korzystaj z dodatkowych bibliotek, wykorzystaj jedynie gniazda. 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 TCP

Gniazda UDP

## 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

### Protokół HTTP w wersji 1.1

- 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,

podsluchaj 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.

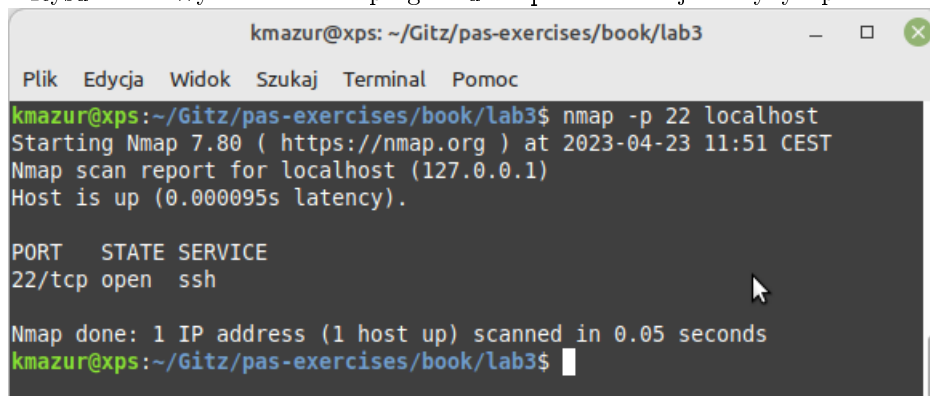
- 7.6** Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/pas-book-p1-ch7-ex6:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 127.0.0.1 na porcie TCP o numerze 7006 działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer udostępnia różne wersje językowe swojej 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.

## 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



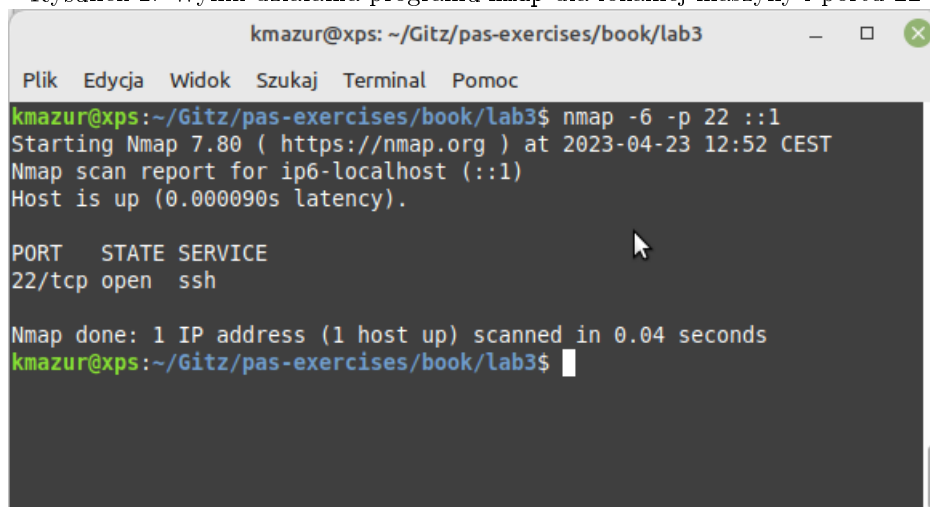
```
kmazur@xps: ~/Gitz/pas-exercises/book/lab3
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
kmazur@xps:~/Gitz/pas-exercises/book/lab3$ nmap -p 22 localhost
Starting Nmap 7.80 ( https://nmap.org ) at 2023-04-23 11:51 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000095s latency).

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.

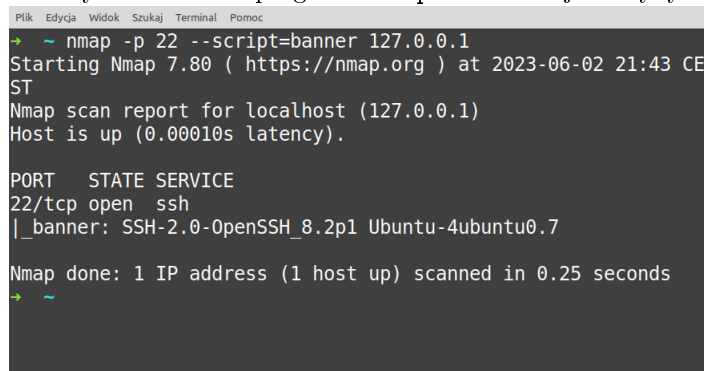
**3.10** Aby przetestować poprawność swojego rozwiązania, możesz skorzystać z gotowego serwera, do którego może połączyć się Twój klient, aby wysłać wiadomość. Aby uruchomić serwer, zainstaluj Dockera, a następnie w konsoli wydaj polecenia:

- 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



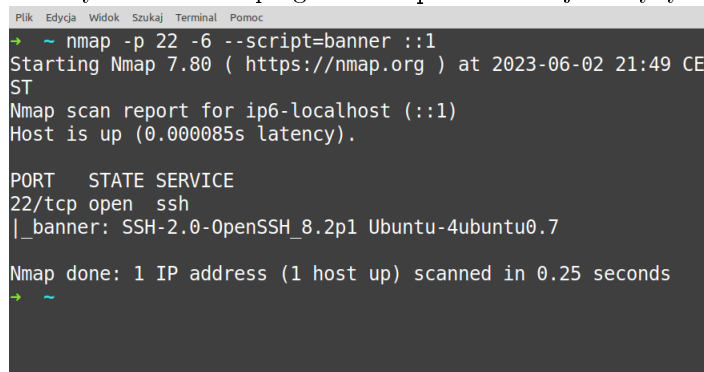
```
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
➔ ~ nmap -p 22 --script=banner 127.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-02 21:43 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00010s latency).

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
➔ ~
```

## Protokoły pocztowe - protokół SMTP

6.1 Do wykonania zadania możesz użyć:

- Klienta `telnet`, jeśli serwer nie wymaga szyfrowania, polecenie do nawiązania połączenia z serwerem: `telnet server_ip port`
- Klienta `OpenSSL`, o nazwie `s_client` jeśli serwer wymaga szyfrowania, polecenie do nawiązania połączenia z serwerem: `openssl s_client -crlf -connect server_ip:port`

Jako serwera SMTP możesz użyć:

- `iRedMail`, który dostępny jest jako kontener Dockerowy,
- Serwera pocztowego udostępnianego np. przez `interia.pl`, gdzie `poczta.interia.pl` jest adresem serwera SMTP, 465 jest numerem portu, na którym działa serwer. Potrzebujesz również konta na serwerze.

Poniżej przykład połączenia z serwerem `poczta.interia.pl`:

```

Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
~ 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
 1 i:C = US, O = DigiCert Inc, CN = DigiCert TLS Hybrid ECC SHA384 2020 CA1
 2 s:C = US, O = DigiCert Inc, CN = DigiCert TLS Hybrid ECC SHA384 2020 CA1
 3 i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIFYzCCB0igAwIBAgIQD/Ztrk8kirkvn+QyffFMbjAKBggqhkJOPQDDAzBWMQsw
CQYDVQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMTAwLGYyVQDDYdEaWdp
Q2VydCBUTFMgSHlicmLkIEVDQyBTSEEzODQgMjAyMCDQTEwHhcNMjIwNzI3MDAw
MDAwWWhcnMjMwNzI3MjI0TU5wBjBiMQswCQYDVQGEwJOTDEPMGA1UEBxMG53Jh
a293MSswCQYDVQKEyJHcnVwYSBjTlRFUKlBLBmIHhNwLlB6IG8uby4gc3AuIGsu
MRUwEwYDVQDDAwQmLudGvYwEuEucGwwWTATBgqhkJOPQIBBggqhkJOPQMBBwNC
-----END CERTIFICATE-----

```

```

Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
0050 - 53 2c b0 aa 04 b3 71 c6-bc a4 d6 e9 b4 f8 3d 77 S,...q.....=W
0060 - 88 f6 2a c2 63 ce de 33-10 2a 43 fc b3 47 ad 54 ..*.c..3.*C..G.T
0070 - 6f 24 48 a1 64 4d 20 28-5c 7f 97 6a 43 e9 d9 0a o$H.dM (\..jC...
0080 - 60 3a 4c 77 b3 4b 9d 62-5d 30 9d 35 3f 5b 41 39 `:Lw.K.b]0.5?[A9
0090 - 1e c9 14 fe cf 62 dd 5d-57 74 3f 4e fc 4f ba 6a ....b.]Wt?N.O.j
00a0 - 6d 6e 09 d9 dd 71 df 1b-27 15 ca f2 b5 15 fe 6e mn...q..'.....n
00b0 - f1 e4 7b c9 97 50 d2 42-be 94 4e be ea 62 31 1d ...{..P.B..N..b1.
00c0 - dc f3 db e5 49 2f ae a8-70 31 cb 3c 6e 31 af 94 ....I/..p1.<n1..

Start Time: 1685967841
Timeout : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
Max Early Data: 0
---
read R BLOCK
220 ESMTPL 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

```





**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.

---

Programowanie aplikacji sieciowych, część 1 | Katarzyna Mazur

---

Programowanie aplikacji sieciowych, część 1 | Katarzyna Mazur
27

```

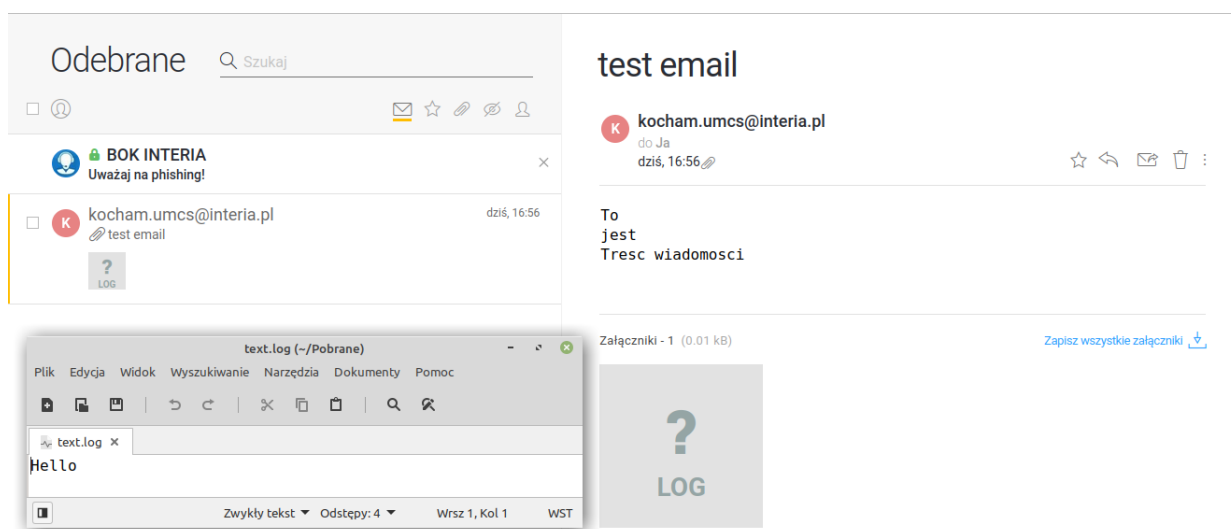
250 CHUNKING
auth login
334 VXNlcm5hbWU6
a29jaGFtLnVtY3NAaW50ZXJpYS5wbA==
334 UGFzc3dvcmQ6
a29jaGFtLnVtY3MyMDIz
235 2.7.0 Authentication successful
mail from: <kocham.umcs@interia.pl>
250 2.1.0 Ok
rcpt to: <pas.umcs@poczta.fm>
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
to: <pas.umcs@poczta.fm>
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
cGFzMjAxN0BpbmRlcmhlLnBs
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>

**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**

**7.7** 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>