

# Programowanie aplikacji sieciowych

Zbiór zadań, część pierwsza

Katarzyna Mazur

## 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>Protokoły pocztowe</b>	<b>12</b>
5.1	Protokół SMTP . . . . .	12
5.2	Protokół POP3 . . . . .	12
5.3	Protokół IMAP . . . . .	12
<b>6</b>	<b>Protokół HTTP</b>	<b>12</b>
<b>7</b>	<b>Odpowiedzi</b>	<b>12</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:

UDP Datagram Header Format								
Bit #	0	7	8	15	16	23	24	31
0	Source Port				Destination Port			
32	Length				Header and Data Checksum			

Napisz program, który z powyższego datagramu UDP wydobydzie:

- 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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**. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). Zadbaj o prawidłową obsługę błędów.

- 3.12** Zmodyfikuj program z zadania **3.11** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). Zadbaj o prawidłową obsługę błędów.
- 3.16** Zmodyfikuj program z zadania **3.15** w taki sposób, aby łączył się z serwerem posiadającym adres IPv6. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). 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. Zadanie rozwiąż bez użycia dodatkowych bibliotek (korzystaj jedynie z gniazd). Zadbaj o prawidłową obsługę błędów.

## 4 Gniazda serwerowe

Gniazda TCP

Gniazda UDP

## 5 Protokoły pocztowe

5.1 Protokół SMTP

5.2 Protokół POP3

5.3 Protokół IMAP

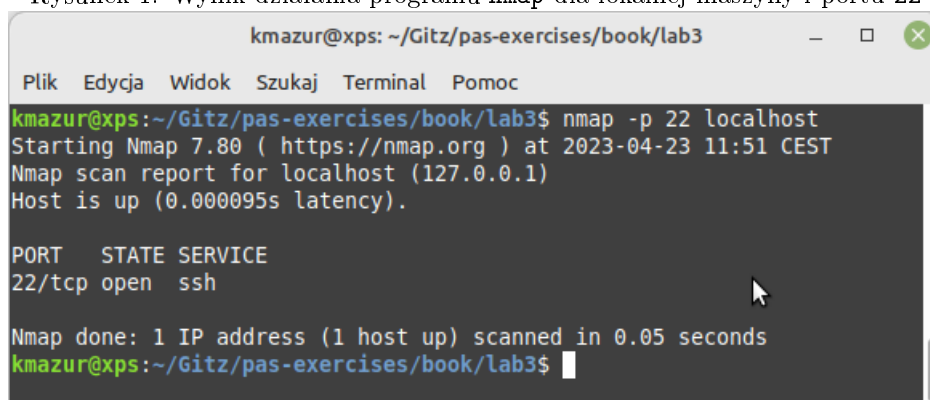
## 6 Protokół HTTP

## 7 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: ~/Git/pas-exercises/book/lab3
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
kmazur@xps:~/Git/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:~/Git/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.
- 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.

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$

```

**3.6** 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
```

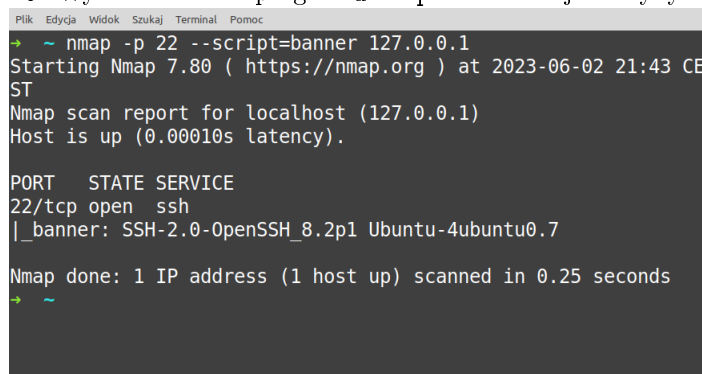
- Uruchom serwer za pomocą Dockera:

```
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



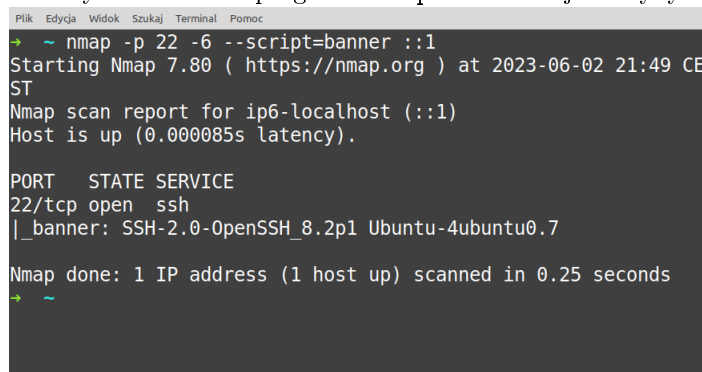
```
Plik  Edycja  Widok  Szukaj  Terminal  Pomoc
→ ~ nmap -p 22 --script=banner 127.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2023-06-02 21:43 CE
ST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00010s latency).

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 CE
ST
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
→ ~
```