SPRAWOZDANIE LABORATORIUM 2.1-2.2 – PROGRAMOWANIE SIECIOWE

Z22 - Bartosz Latosek, Adam Sudoł, Karol Rogoziński, Bartłomiej Dudek

2.1

Funkcjonalny serwer TCP w Pythonie po zbudowaniu oraz jego współpraca z klientem C:

```
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Server$ bash build.sh
Sending build context to Docker daemon  5.632kB
Step 1/3 : FROM python:3
 ---> 00cd1fb8bdcc
Step 2/3 : ADD z22_server_1_tcp.py /
 ---> Using cache
 ---> e4f73fac4d7e
Step 3/3 : ENTRYPOINT ["python3", "./z22_server_1_tcp.py"]
 ---> Using cache
 ---> 49867df96469
Successfully built 49867df96469
Successfully tagged z22_server_1_tcp_py:latest
```

```
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Server$ bash run.sh
Server will run on 172.21.22.5:9000
Message received from client: Message sent from c client
```

Funkcjonalny klient TCP w C po zbudowaniu oraz jego współpraca z serwerem:

```
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/C/Client$ ./build.sh
Sending build context to Docker daemon  6.144kB
Step 1/4 : FROM gcc:4.9
 ---> 1b3de68a7ff8
Step 2/4 : COPY . /
 ---> Using cache
 ---> 3f46bdd261df
Step 3/4 : RUN gcc -o z22_client_1_tcp z22_client_1_tcp.c
 ---> Using cache
 ---> aad0dfdeb3b9
Step 4/4 : ENTRYPOINT ["./z22_client_1_tcp"]
 ---> Using cache
 ---> 18e3801c9cc2
Successfully built 18e3801c9cc2
Successfully tagged z22_client_1_tcp_c:latest
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/C/Client$ ./run.sh
Setting default port to 9000
Message received from server: Message received.
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/C/Client$
```

Funkcjonalny serwer TCP w C po zbudowaniu oraz jego współpraca z klientem Pythonie:

```
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/C/Server$ bash build.sh
Sending build context to Docker daemon  6.656kB
Step 1/4 : FROM gcc:4.9
 ---> 1b3de68a7ff8
Step 2/4 : COPY . /
 ---> Using cache
 ---> a46f0b37fc39
Step 3/4 : RUN gcc -o z22_server_1_tcp z22_server_1_tcp.c
 ---> Using cache
 ---> 6dda9c7207b2
Step 4/4 : ENTRYPOINT ["./z22_server_1_tcp"]
 ---> Using cache
 ---> 4a213045bdf6
Successfully built 4a213045bdf6
Successfully tagged z22_server_1_tcp_c:latest
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/C/Server$ bash run.sh
Setting default port to 9000
C Server listening on 172.21.22.5:9000
Connection from 172.21.22.2
Message received from client: Top secret
```

Funkcjonalny klient TCP w Pythonie po zbudowaniu oraz jego współpraca z serwerem w C:

```
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Client$ bash build.sh
Sending build context to Docker daemon   5.12kB
Step 1/3 : FROM python:3
 ---> 00cd1fb8bdcc
Step 2/3 : ADD /z22_client_1_tcp.py /
 ---> Using cache
 ---> c5c802cf5645
Step 3/3 : ENTRYPOINT ["python3", "./z22_client_1_tcp.py"]
 ---> Using cache
 ---> ae12a2528ed3
Successfully built ae12a2528ed3
Successfully tagged z22_client_1_tcp_py:latest
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Client$ bash run.sh
Client connecting to 172.21.22.5:9000
Data received from server: Message received.
Client finished.
```

2.2

Funkcjonalny serwer TCP w C z buforem mniejszym od jednorazowo wysyłanej przez klienta porcji danych po zbudowaniu oraz jego współpraca z klientem w Pythonie:

```
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/C/Server_2$ bash build.sh
Sending build context to Docker daemon  6.656kB
Step 1/4 : FROM gcc:4.9
 ---> 1b3de68a7ff8
Step 2/4 : COPY . /
 ---> Using cache
 ---> da1a9f6bdc43
Step 3/4 : RUN gcc -o z22_server_2_tcp z22_server_2_tcp.c
 ---> Using cache
 ---> 9eb8c68a0022
Step 4/4 : ENTRYPOINT ["./z22_server_2_tcp"]
 ---> Using cache
 ---> 4c87e914d581
Successfully built 4c87e914d581
Successfully tagged z22_server_2_tcp_c:latest
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/C/Server_2$ bash run.sh
Setting default port to 9000
C Server listening on 172.21.22.5:9000
Connection from 172.21.22.2
> 5 : Hello
> 5 :  worl
> 3 : d!0
```

```
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Client_2$ bash build.sh
Sending build context to Docker daemon   5.12kB
Step 1/3 : FROM python:3
 ---> 00cd1fb8bdcc
Step 2/3 : ADD /z22_client_2_tcp.py /
 ---> Using cache
 ---> 2d0c6b52bf22
Step 3/3 : ENTRYPOINT ["python3", "./z22_client_2_tcp.py"]
 ---> Using cache
 ---> a6bc047f7a7f
Successfully built a6bc047f7a7f
Successfully tagged z22_client_2_tcp_py:latest
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Client_2$ bash run.sh
Client connecting to 172.21.22.5:9000
Data received from server: Message received.
Client finished.
```

Funkcjonalny serwer TCP w Pythonie z buforem mniejszym od jednorazowo wysyłanej przez klienta porcji danych oraz jego współpraca z klientem w Pythonie:

```
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Server_2$ bash run.sh
Server will run on 172.21.22.5:9000
Received data: abcdef0
```

```
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Client_2$ bash build.sh
Sending build context to Docker daemon   5.12kB
Step 1/3 : FROM python:3
 ---> 00cd1fb8bdcc
Step 2/3 : ADD /z22_client_2_tcp.py /
 ---> Using cache
 ---> 2d0c6b52bf22
Step 3/3 : ENTRYPOINT ["python3", "./z22_client_2_tcp.py"]
 ---> Using cache
 ---> a6bc047f7a7f
Successfully built a6bc047f7a7f
Successfully tagged z22_client_2_tcp_py:latest
bdudek@bigubu:~/lab_1/psi_22l_KK_z22/PSI_2/Python/Client_2$ bash run.sh
Client connecting to 172.21.22.5:9000
Data received from server: Message received.
Client finished.
```

Różnica między send(), a sendall():

Funkcja send() w Pythonie nie musi przesłać wszystkich danych, które zostaną jej przekazane podczas, gdy funkcja sendall() nie zakończy swojego działania do momentu wysłania całych otrzymanych danych, co może zająć więcej czasu.

Modyfikacje programów:

Serwer bez zmian w obsłudze względem większego bufora otrzymuje tylko część wiadomości po czym blokuje się:

```
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/Python/Server_2$ ./run.sh
Server will run on 172.21.22.5:9000
Message received from client: Me
```

Przyczyną tego zdarzenia jest blokada nałożona przez funkcję recv(), która oczekuje na kolejne dane.

Jedną z możliwości rozwiązania tego problemu jest uprzednie wysyłanie długości bufora, a potem danych:

```
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/Python/Server_2$ ./run.sh
Server will run on 172.21.22.5:9000
Received data length: 7
Message received from client: Bazinga
```

```
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/Python/Client_2$ ./run.sh
Client connecting to 172.21.22.5:9000
Data length: 7
Data received from server: Message received.
Client finished.
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/Python/Client_2$
```

Kod tego rozwiązania:

Klient:

```
with s.socket(s.AF_INET, s.SOCK_STREAM) as client:
  client.connect((HOST, PORT));
  data = random.choice(messages)

  data_len = len(data)
  print(f"Data length: {len(data)}")
  client.sendall(str(data_len).encode('utf-8'))
  client.sendall(data.encode('utf-8'))
  data = client.recv(BUFFER)
  print(f"Data received from server: {data.decode('utf-8')}")
  client.close()

print("Client finished.")
```

Serwer:

```
with s.socket(s.AF_INET, s.SOCK_STREAM) as server:
  server.bind((HOST, PORT))
  server.listen( 5 )
  print(f"Server will run on {HOST}:{PORT}")
  while True:
    client, address = server.accept()
    with client:
      msg_len = client.recv(BUFFER_FOR_LEN)
      msg_len = int(msg_len.decode('utf-8'))
      print(f"Received data length: {msg_len}")
      data = client.recv(msg_len)
      print(f"Message received from client: {data.decode('utf-8')}")
      client.sendall("Message received.".encode('utf-8'))
    client.close()
```

Jego wadą jest wymaganie określenia maksymalnej długości wysyłanej wiadomości z góry.

Kolejnym rozwiązaniem tego problemu jest skorzystanie z settimeout(), aby zamknąć połączenie jeśli serwer nie otrzyma danych przez długi okres czasu.

Kod serwera:

```
while True:
  client, address = server.accept()
  client.settimeout(2.0)
  data = b''
  while True:
    part = client.recv(BUFFER)
    data += part
    if len(part) < BUFFER:
      break
  client.settimeout(None)
  print(f"Received data: {data.decode('utf-8')}")
  client.sendall("Message received.".encode('utf-8'))
  client.close()
```

Główny problem tego rozwiązania występuje, kiedy długość przesyłanej wiadomości jest wielokrotnością długości bufora, co powoduje zawieszenie gniazda przez funkcję recv() do czasu nastąpienia timeout'u. Rozwiązaniem tego problemu może być zastosowanie znaku specjalnego na końcu wiadomości, który rozpozna serwer.

Kod rozwiązania:

```
while True:
  client, address = server.accept()
  client.settimeout(2.0)
  data = b''
  while True:
    part = client.recv(BUFFER)
    data += part
    if "0" in part.decode('utf-8'):
      break
  client.settimeout(None)
  print(f"Received data: {data.decode('utf-8')}")
  client.sendall("Message received.".encode('utf-8'))
  client.close()
```

To samo rozwiązanie dla C:

Server:

```
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/C/Server_2$ ./run.sh
Setting default port to 9000
C Server listening on 172.21.22.5:9000
Connection from 172.21.22.2
> 5 : Messa
> 5 : ge se
> 5 : nt fr
> 5 : om c
> 5 : clien
> 1 : t
```

Klient:

```
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/C/Client$ ./run.sh
Setting default port to 9000
Message received from server: Message received.
blatosek@bigubu:~/psi_22l_KK_z22/PSI_2/C/Client$
```