

Programowanie sieciowe

zadanie 1.1

12.11.2024

Bogumił Stoma

Aleksander Stanoch

Sebastian Abramowski

Treść zadania

Z 1 Komunikacja UDP

Napisz zestaw dwóch programów – klienta i serwera wysyłające datagramy UDP. Wykonaj ćwiczenie w kolejnych inkrementalnych wariantach (rozszerzając kod z poprzedniej wersji). Klient jak i serwer powinien być napisany zarówno w C jak i Pythonie (4 programy). Sprawdzić i przetestować działanie „między-platformowe”, tj. klient w C z serwerem Python i vice versa.

Z 1.1

Klient wysyła, a serwer odbiera datagramy o stałym rozmiarze (rzędu kilkuset bajtów). Datagramy powinny posiadać ustaloną formę danych. Przykładowo: pierwsze dwa bajty datagramu mogą zawierać informację o jego długości, a kolejne bajty kolejne litery A-Z powtarzające się wymaganą liczbę razy (ale można przyjąć inne rozwiązanie). Odbiorca powinien weryfikować odebrany datagram i odsyłać odpowiedź o ustalonej formie. Klient powinien wysyłać kolejne datagramy o przyrastającej wielkości np. 1, 100, 200, 1000, 2000... bajtów. Sprawdzić, jaki był maksymalny rozmiar wysłanego (przyjętego) datagramu. Ustalić z dokładnością do jednego bajta jak duży datagram jest obsługiwany. Wyjaśnić.

Rozwiązanie zadania

Zadanie polegało na stworzeniu klienta i serwera, które komunikują się za pomocą protokołu UDP, wysyłając datagramy o rosnących rozmiarach, aż do napotkania błędu. W miarę rosnących rozmiarów datagramów (w naszym przypadku o 1000 bajtów), klient wysyła wiadomości, a serwer je odbiera i odpowiada potwierdzeniem. Kiedy wysłanie wiadomości o określonej wielkości powoduje wyjątek, klient zmniejsza rozmiar datagramu o 1000 bajtów i próbuje go zwiększać co 1 bajt, aż do kolejnego błędu. Dzięki temu możliwe jest dokładne określenie maksymalnego rozmiaru datagramu obsługiwanego przez połączenie.

Implementacja w Pythonie

W Pythonie implementacja była stosunkowo prosta dzięki bibliotece `socket`, która udostępnia funkcje `sendto` i `recvfrom` dla protokołu UDP. Stworzyliśmy pomocnicze funkcje, aby przygotować datagram, tak jak polecono w poleceniu, pierwsze dwa bajty zawierają rozmiar wiadomości, a kolejne bajty to powtarzający się alfabet. Odbiór i nadawanie odbywały się na tych samych gniazdach zarówno w kliencie, jak i serwerze. Dzięki automatycznemu zarządzaniu pamięcią, tworzenie i zwalnianie gniazd nie wymagało specjalnej uwagi.

Implementacja w C

W C implementacja również wykorzystuje funkcje `sendto` i `recvfrom` do wysyłania i odbierania datagramów, ale wymaga ręcznego zarządzania pamięcią i gniazdami. Tworzenie gniazd było bardziej skomplikowane niż w pythonie, natomiast w ostateczności było bardzo podobne.

Opis konfiguracji testowej

- **Adres IP serwera:** `172.21.35.2` (może być przekazany jako argument w linii poleceń dla klienta C i Pythona).
- **Port:** `12345` (domyślnie, ale można zmienić, podając go jako argument w linii poleceń).

Adres pierwszego utworzonego kontenera w sieci naszego zespołu (172.21.35.2):

```
sabramow@bigubu: ~/psi-lab1/client
abramow@bigubu:~/psi-lab1/client$ docker network inspect z35_network | grep -C 5 IPv4
    "Containers": {
      "a2ea4fd318eb53eea7b19025597749a9da5e380636c2bdfad50807686f59a5fe": {
        "Name": "z35_pserver_container",
        "EndpointID": "30f8c55676b23a847dd1ae1849d6561ca2fbb476ceb320c3553d92c9f0e41021",
        "MacAddress": "02:42:ac:15:23:02",
        "IPv4Address": "172.21.35.2/24",
        "IPv6Address": "fd00:1032:ac21:35::2/64"
      }
    },
    "Options": {},
    "Labels": {}
  }
abramow@bigubu:~/psi-lab1/client$
```

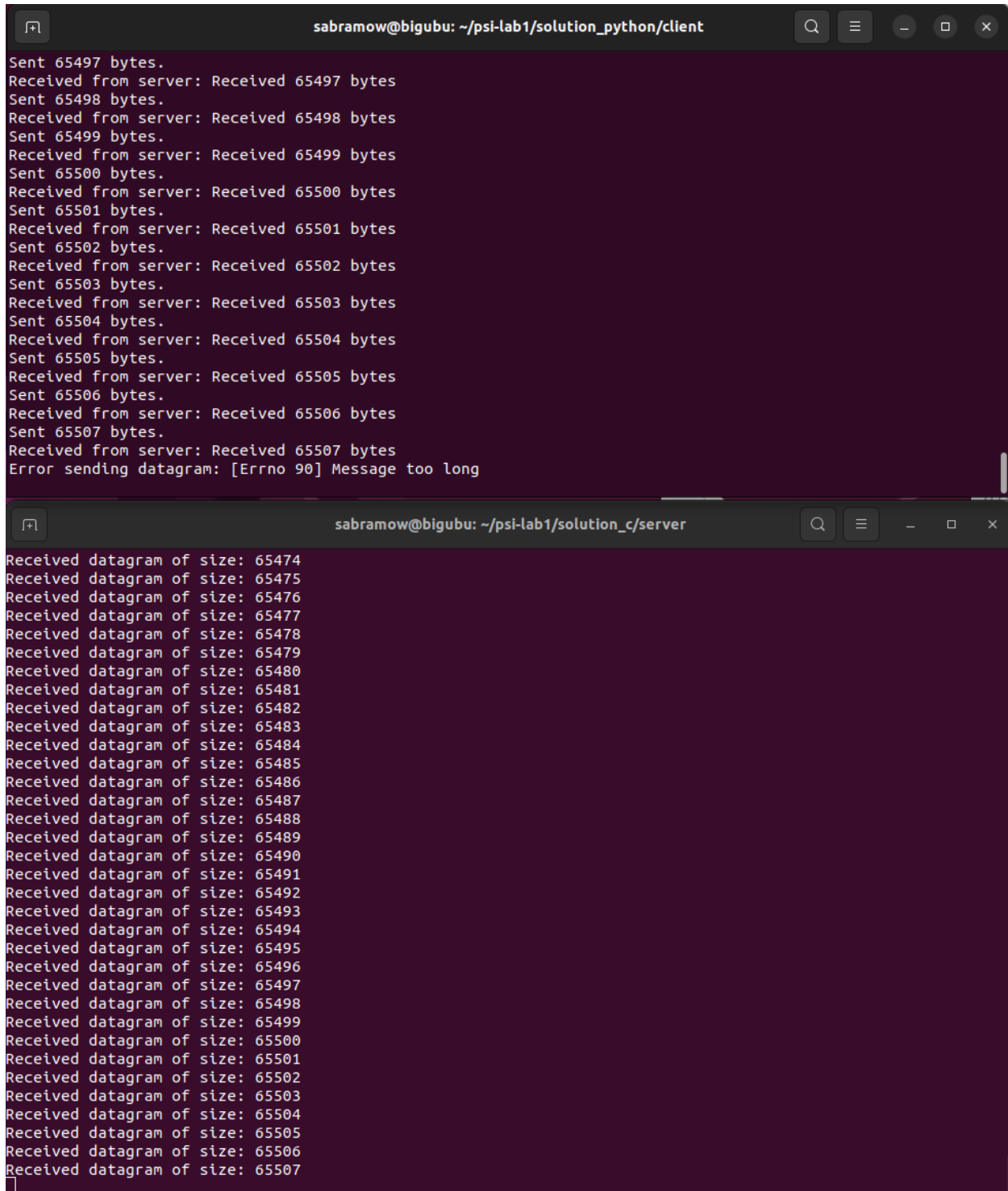
Aby uruchomić kontenery, należało utworzyć odpowiedni plik Dockerfile, zbudować obraz (image) i uruchomić go, zwracając uwagę na właściwe ustawienie nazw obrazu, kontenera oraz sieci.

Poniżej przykład ręcznego uruchomienia jednego z kontenerów.

```
sabramow@bigubu: ~/psi-lab1/client
sabramow@bigubu:~/psi-lab1/client$ docker rm z35_pclient_container
z35_pclient_container
sabramow@bigubu:~/psi-lab1/client$ docker build -t z35_pclient .
[+] Building 2.9s (7/7) FINISHED
=> [internal] load build definition from dockerfile                                docker:default
=> => transferring dockerfile: 105B                                              0.1s
=> [internal] load metadata for docker.io/library/python:3                      0.0s
=> [internal] load .dockerignore                                                1.1s
=> => transferring context: 2B                                                  0.1s
=> [internal] load build context                                                0.0s
=> => transferring context: 32B                                                 0.1s
=> [1/2] FROM docker.io/library/python:3@sha256:a31cbb4db18c6f09e3300fa85b77f6d56702501fcb9bdb8 0.0s
=> CACHED [2/2] COPY ./clientP.py ./                                          0.0s
=> exporting to image                                                         0.2s
=> => exporting layers                                                         0.0s
=> => writing image sha256:e7fe885839f61f87db7878a33df43c64f059ffe32a3ae708789e670adcc107ca 0.1s
=> => naming to docker.io/library/z35_pclient                                0.1s
sabramow@bigubu:~/psi-lab1/client$ docker run -it --network z35_network --name z35_pclient_container z35_pclient
Sent 1002 bytes.
```

Przykłady działania

Poniżej testy z serwerem w języku C oraz klientem w pythonie (uruchomienie serwera w pythonie i klienta w C działa naturalnie tak samo)



The image displays two terminal windows side-by-side, illustrating a network communication test. The top window, titled 'sabramow@bigubu: ~/psi-lab1/solution_python/client', shows the output of a Python client. It displays a series of 'Sent' and 'Received from server' messages for data sizes ranging from 65497 to 65507 bytes. The final line indicates an error: 'Error sending datagram: [Errno 90] Message too long'. The bottom window, titled 'sabramow@bigubu: ~/psi-lab1/solution_c/server', shows the output of a C server. It displays a series of 'Received datagram of size' messages for data sizes ranging from 65474 to 65507 bytes, corresponding to the data sent by the client.

```
sabramow@bigubu: ~/psi-lab1/solution_python/client
Sent 65497 bytes.
Received from server: Received 65497 bytes
Sent 65498 bytes.
Received from server: Received 65498 bytes
Sent 65499 bytes.
Received from server: Received 65499 bytes
Sent 65500 bytes.
Received from server: Received 65500 bytes
Sent 65501 bytes.
Received from server: Received 65501 bytes
Sent 65502 bytes.
Received from server: Received 65502 bytes
Sent 65503 bytes.
Received from server: Received 65503 bytes
Sent 65504 bytes.
Received from server: Received 65504 bytes
Sent 65505 bytes.
Received from server: Received 65505 bytes
Sent 65506 bytes.
Received from server: Received 65506 bytes
Sent 65507 bytes.
Received from server: Received 65507 bytes
Error sending datagram: [Errno 90] Message too long

sabramow@bigubu: ~/psi-lab1/solution_c/server
Received datagram of size: 65474
Received datagram of size: 65475
Received datagram of size: 65476
Received datagram of size: 65477
Received datagram of size: 65478
Received datagram of size: 65479
Received datagram of size: 65480
Received datagram of size: 65481
Received datagram of size: 65482
Received datagram of size: 65483
Received datagram of size: 65484
Received datagram of size: 65485
Received datagram of size: 65486
Received datagram of size: 65487
Received datagram of size: 65488
Received datagram of size: 65489
Received datagram of size: 65490
Received datagram of size: 65491
Received datagram of size: 65492
Received datagram of size: 65493
Received datagram of size: 65494
Received datagram of size: 65495
Received datagram of size: 65496
Received datagram of size: 65497
Received datagram of size: 65498
Received datagram of size: 65499
Received datagram of size: 65500
Received datagram of size: 65501
Received datagram of size: 65502
Received datagram of size: 65503
Received datagram of size: 65504
Received datagram of size: 65505
Received datagram of size: 65506
Received datagram of size: 65507
```

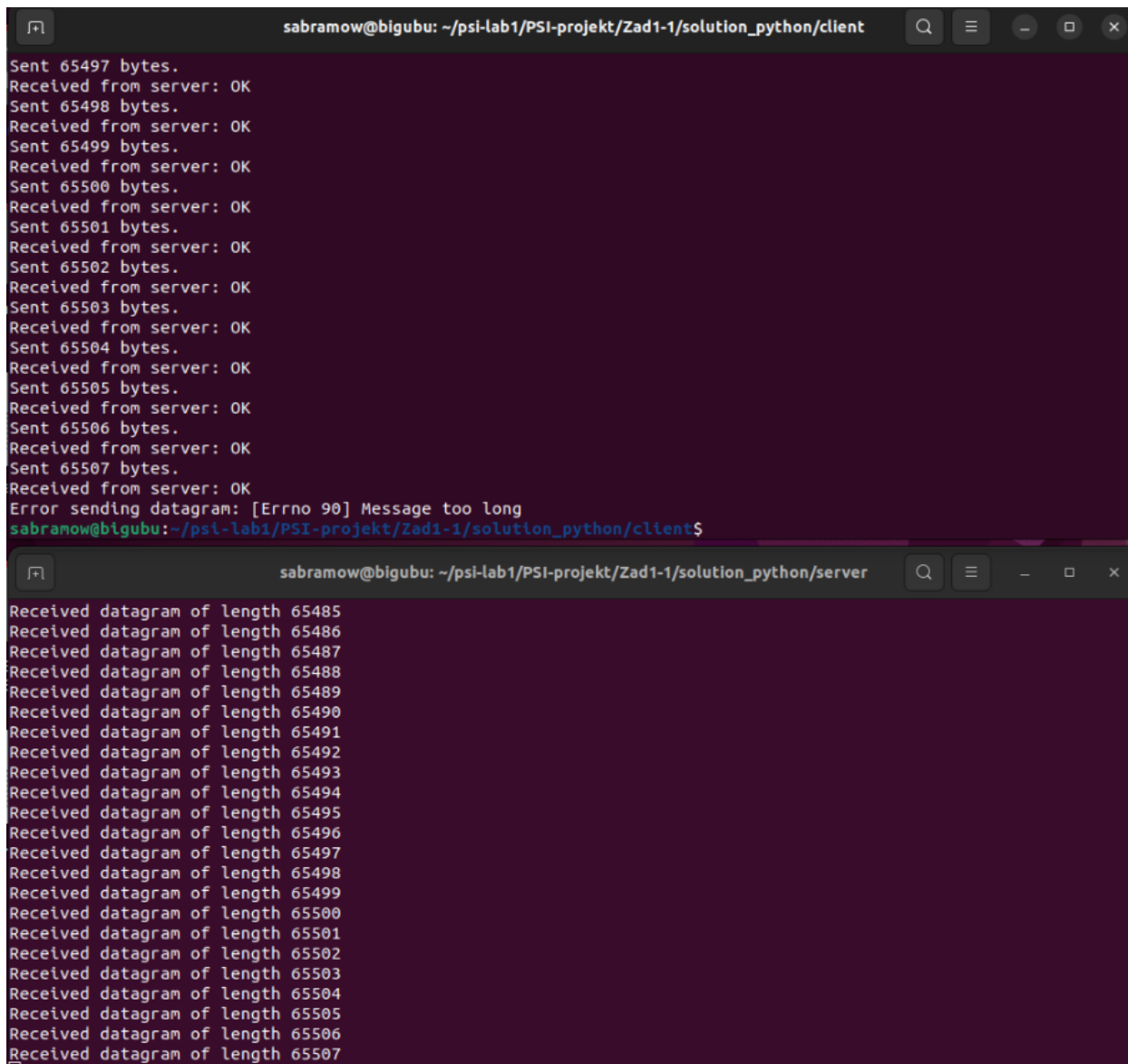
Uruchomienie serwera w C

```
bstoma@bigubu:~/PSI-zad-1-1/Zad1-1/solution_c/server$ ./run.sh
Starting server C
[+] Building 9.8s (8/8) FINISHED                                docker:default
=> [internal] load build definition from dockerfile              0.3s
=> => transferring dockerfile: 114B                             0.0s
=> [internal] load metadata for docker.io/library/gcc:4.9       2.2s
=> [internal] load .dockerignore                                0.5s
=> => transferring context: 2B                                    0.0s
=> [internal] load build context                                0.9s
=> => transferring context: 2.58kB                                0.0s
CACHED [1/3] FROM docker.io/library/gcc:4.9@sha256:6356ef8b29cc3522527a85b6c58a28626744514bea87a10ff2bf67599a 0.5
=> resolve docker.io/library/gcc:4.9@sha256:6356ef8b29cc3522527a85b6c58a28626744514bea87a10ff2bf67599a7474f5 0.5
[2/3] COPY ./ ./                                               0.8s
=> [3/3] RUN gcc -o server serverC.c                             2.1s
=> exporting to image                                           1.1s
=> => exporting layers                                           0.6s
=> => writing image sha256:ecf4de1011935b78c0f7b7bd217aafdf5c418cdc96291f0b18afaa73a606c570 0.0s
=> => naming to docker.io/library/z35_cserver                   0.1s
Server is listening on port 12345
Received datagram of size: 1000
```

Uruchomienie klienta w C

```
bstoma@bigubu: ~/PSI-zad-1-1 x bstoma@bigubu: ~/PSI-zad-1 x + v
Sent datagram of size: 65479, received response: Received 65479 bytes
Sent datagram of size: 65480, received response: Received 65480 bytes
Sent datagram of size: 65481, received response: Received 65481 bytes
Sent datagram of size: 65482, received response: Received 65482 bytes
Sent datagram of size: 65483, received response: Received 65483 bytes
Sent datagram of size: 65484, received response: Received 65484 bytes
Sent datagram of size: 65485, received response: Received 65485 bytes
Sent datagram of size: 65486, received response: Received 65486 bytes
Sent datagram of size: 65487, received response: Received 65487 bytes
Sent datagram of size: 65488, received response: Received 65488 bytes
Sent datagram of size: 65489, received response: Received 65489 bytes
Sent datagram of size: 65490, received response: Received 65490 bytes
Sent datagram of size: 65491, received response: Received 65491 bytes
Sent datagram of size: 65492, received response: Received 65492 bytes
Sent datagram of size: 65493, received response: Received 65493 bytes
Sent datagram of size: 65494, received response: Received 65494 bytes
Sent datagram of size: 65495, received response: Received 65495 bytes
Sent datagram of size: 65496, received response: Received 65496 bytes
Sent datagram of size: 65497, received response: Received 65497 bytes
Sent datagram of size: 65498, received response: Received 65498 bytes
Sent datagram of size: 65499, received response: Received 65499 bytes
Sent datagram of size: 65500, received response: Received 65500 bytes
Sent datagram of size: 65501, received response: Received 65501 bytes
Sent datagram of size: 65502, received response: Received 65502 bytes
Sent datagram of size: 65503, received response: Received 65503 bytes
Sent datagram of size: 65504, received response: Received 65504 bytes
Sent datagram of size: 65505, received response: Received 65505 bytes
Sent datagram of size: 65506, received response: Received 65506 bytes
Sent datagram of size: 65507, received response: Received 65507 bytes
sendto failedbstoma@bigubu:~/PSI-zad-1-1/Zad1-1/solution_c/client$ |
```

Uruchomienie serwera i klienta w pythonie



```
sabramow@bigubu: ~/psi-lab1/PSI-projekt/Zad1-1/solution_python/client
Sent 65497 bytes.
Received from server: OK
Sent 65498 bytes.
Received from server: OK
Sent 65499 bytes.
Received from server: OK
Sent 65500 bytes.
Received from server: OK
Sent 65501 bytes.
Received from server: OK
Sent 65502 bytes.
Received from server: OK
Sent 65503 bytes.
Received from server: OK
Sent 65504 bytes.
Received from server: OK
Sent 65505 bytes.
Received from server: OK
Sent 65506 bytes.
Received from server: OK
Sent 65507 bytes.
Received from server: OK
Error sending datagram: [Errno 90] Message too long
sabramow@bigubu: ~/psi-lab1/PSI-projekt/Zad1-1/solution_python/client$

sabramow@bigubu: ~/psi-lab1/PSI-projekt/Zad1-1/solution_python/server
Received datagram of length 65485
Received datagram of length 65486
Received datagram of length 65487
Received datagram of length 65488
Received datagram of length 65489
Received datagram of length 65490
Received datagram of length 65491
Received datagram of length 65492
Received datagram of length 65493
Received datagram of length 65494
Received datagram of length 65495
Received datagram of length 65496
Received datagram of length 65497
Received datagram of length 65498
Received datagram of length 65499
Received datagram of length 65500
Received datagram of length 65501
Received datagram of length 65502
Received datagram of length 65503
Received datagram of length 65504
Received datagram of length 65505
Received datagram of length 65506
Received datagram of length 65507
```

Wnioski końcowe i uwagi własne

Testy wykazały, że komunikacja UDP działa zgodnie z oczekiwaniami, a maksymalny rozmiar obsługiwanego datagramu wynosi około 65,507 bajtów, bo 65,535 bajtów (rozmiar całego pakietu IP) - 8 bajtów (nagłówek UDP) - 20 bajtów (nagłówek IP) = 65,507 bajtów. Należy pamiętać, że różne systemy operacyjne oraz konfiguracje sieciowe mogą wpłynąć na wydajność i maksymalny rozmiar datagramu.