PSI 23Z Lab - Sprawozdanie z zadań 2.x

Zespół 21 w składzie:

- Damian Pałyska
- Michał Bogiel
- Jan Kowalczewski
- Mariusz Pakulski

Data: w nagłówku

Wersja: 1.

Polecenie (wariant Aa)

Zadanie 1 Komunikacja TCP

Napisz zestaw dwóch programów – klienta i serwera komunikujących się poprzez TCP. Wykonaj ćwiczenie w kolejnych inkrementalnych wariantach (rozszerzając kod z poprzedniej wersji).

Platforma testowa

Do testów wykorzystujemy kontenery Docker uruchamiane przez plik docker-compose. W każdym zadaniu występują dwa kontenery:

- Kontener z_21_2x_client dla klienta bazujący na gcc w wersji 4.9
- Kontener z_21_2x_server dla serwera bazujący na obrazie python w wersji 3

Kontenery komunikują się między sobą w sieci z21_network, czyli 172.21.21.0/24.

Wszystkie testy uruchamiane były na serwerze bigubu.ii.pw.edu.pl i stamtąd pochodzą wszystkie wydruki.

Zadanie 2.1

Polecenie

Klient wysyła, a serwer odbiera „złożoną” strukturę danych o stałym rozmiarze (rzędu kilkuset bajtów). Datagramy powinny posiadać ustalony format danych: pierwsze cztery bajty zawierają numer kolejny pakietu (liczony od 0, typ int32), kolejne dwa bajty datagramu powinny zawierać informację o jego długości (typ int16), a kolejne bajty to kolejne drukowalne znaki powtarzające się wymaganą liczbę razy, aby osiągnąć zakładany rozmiar. Odbiorca powinien weryfikować odebrany datagram i odsyłać odpowiedź (potwierdzenie) o ustalonym formacie. Serwer powinien sygnalizować brak pakietu (przeskok w numeracji pakietu). Może być pomocne użycie pakietu struct w Python do konstrukcji i odbierania danych w zadanym formacie.

Klient w C, serwer w Python.

Można także napisać wersje klienta i serwera w obu językach, ale muszą ze sobą współpracować. Należy zwrócić uwagę na rozmiary danych odczytanych z funkcji sieciowych i weryfikować z rozmiarem przesłanym w „nagłówku” danych. Sygnalizować rozbieżność.

Opis rozwiązania

Klient (C)

```
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock == -1) bailout("opening stream socket");

server.sin_family = AF_INET;
hp = gethostbyname(argv[1]);
if (hp == NULL) errx(2, "%s: unknown host\n", argv[1]);

memcpy(&server.sin_addr, hp->h_addr, hp->h_length);
server.sin_port = htons(atoi(argv[2]));

if (connect(sock, (struct sockaddr *) &server, sizeof(server)) == -1)
    bailout("connecting stream socket");

num_packets = atoi(argv[3]);
delay_ms = atoi(argv[4]);

for (packet_number = 0; packet_number < num_packets; packet_number++)
{
    int32_t packet_number_net = htonl(packet_number);
    int16_t data_length_net = htons(BSIZE - 6);
    memcpy(buffer, &packet_number_net, sizeof(packet_number_net));
    memcpy(buffer + sizeof(packet_number_net), &data_length_net, sizeof(data_length_net));

    for (int i = 6; i < BSIZE; i++)
    {
        buffer[i] = current_char;
        current_char++;
        if (current_char > ASCII_END) current_char = ASCII_START;
    }

    if (write(sock, buffer, BSIZE) == -1) bailout("writing on stream socket");

    // Odbiór i weryfikacja odpowiedzi od serwera
    if (read(sock, recv_buffer, BSIZE) == -1) bailout("reading from stream socket");

    // Wyświetlanie komunikatu o błędzie (jeśli wystąpił)
    if (strncmp(recv_buffer, "OK", 2) != 0) {
        printf("Received error response from server: %s\n", recv_buffer);
    } else {
        printf("Sent packet %d and received confirmation\n", packet_number);
    }

    usleep(delay_ms * 1000);  // opóźnienie
}

close(sock);
```

Klient tworzy gniazdo oraz konfiguruje i nawiązuje połączenie z serwerem (za pomocą funkcji *connect*). W głównej pętli tworzy datagramy zawierające:

- Kolejny numer pakietu (4 bajty, int32; jest on później inkrementowany), Długość
- datagramu (2 bajty, int16).
- Dane w postaci drukowalnych znaków ASCII. Klient wysyła te datagramy do serwera i oczekuje na potwierdzenie odbioru.

Każdy pakiet zostaje wysłany za pomocą systemowej funkcji *write* (która pisze do gniazda strumienia TCP). Następnie czeka na potwierdzenie weryfikacji od serwera (czyta z gniazda strumienia TCP za pomocą systemowej funkcji *read*). Weryfikuje potwierdzenie, aby sprawdzić czy wszystko jest ok. (porównanie napisów za pomocą *strncmp*). Jeśli nie jest, to wyświetla otrzymany komunikat o błędzie. Następnie inkrementuje numer następnego pakietu, zasypia na chwilę i kontynuuje działanie od nowa.

Serwer (Python)

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen(1)
        conn, addr = s.accept()
        with conn:
            print('Connected by', addr)
            expected_packet_number = 0

            while True:
                data = conn.recv(BUFSIZE)
                if not data:
                    break

                packet_number, data_length = struct.unpack('!Ih', data[:6])
                message_data = data[6:6+data_length]
                received_length = len(data)

                if packet_number != expected_packet_number:
                    print(f"Missing packet! Expected {expected_packet_number}, received
{packet_number}")
                    conn.sendall(b'Missing packet')
                    expected_packet_number = packet_number + 1
                    continue

                if received_length != data_length + 6:
                    print(f"Incorrect data length. Expected {data_length + 6}, received
{received_length}")
                    conn.sendall(b'Incorrect data length')
                    continue

                if not is_printable_ascii(message_data):
                    print("Data is not printable ASCII")
                    conn.sendall(b'Data is not printable ASCII')
                    continue

                print(f"\nReceived packet # {packet_number}, length: {received_length}, content:
\n{message_data.decode('ascii', errors='ignore')}")
                conn.sendall(b'OK')
                expected_packet_number += 1
```

Serwer tworzy gniazdo powiązane z określonym hostem i portem (metoda bind) i nasłuchuje (*listen*), oczekując na nawiązanie połączenia (*accept* – ta metoda zwraca połączenie i adres) . Odbiera datagramy (*conn.recv*) i sprawdza ich poprawność pod kątem:

- Numeru kolejnego pakietu (dla wykrycia ewentualnych utrat pakietów).
- Długości danych (porównanie z deklarowaną długością w datagramie).
- Poprawności danych (czy są drukowalnymi znakami ASCII – is_printable_ascii).

Po każdym otrzymanym datagramie, jeśli wszystko się zgadza, serwer odsyła potwierdzenie odbioru do klienta (i zwiększa spodziewany numer następnego pakietu). Jeśli coś się nie zgadza, serwer przesyła komunikat o błędzie. Komunikacja ta odbywa się za pomocą *conn.sendall*.

Testowanie

Testowanie polegało na uruchomieniu serwera i klienta, a następnie obserwowaniu logów serwera podczas odbierania i przetwarzania datagramów.

```
z21_11_server  | Will listen on  0.0.0.0 : 8888 z21_11_client
| address resolved...
z21_11_client  | Resolved IP: 172.21.21.2 z21_11_client  |
Sent packet 0 and received confirmation z21_11_server  |
z21_11_server  | Received packet # 0, length: 512, content:
z21_11_server  |  !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>  z21_11_server   |  Sending
confirmation for packet # 0 z21_11_server  | z21_11_server
| z21_11_server  | Received packet # 1, length: 512, content:
z21_11_server  | ?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]
z21_11_server   |  Sending confirmation for packet # 1
z21_11_server  | z21_11_client  | Sent packet 1 and received
confirmation z21_11_server  | z21_11_server  | Received packet
# 2, length: 512, content:
z21_11_server  | ^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
```

```
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|
z21_11_server    |  Sending  confirmation  for  packet  #  2
z21_11_server  | z21_11_client  | Sent packet 2 and received
confirmation  z21_11_client   |  Sent  packet  3  and  received
confirmation z21_11_server  | z21_11_server  | Received packet
# 3, length: 512, content: z21_11_server  | }~ !"#$%&'()*+,-
./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<   z21_11_server     |   Sending
confirmation for packet # 3 z21_11_server  | z21_11_client
| Sent packet 4 and received confirmation z21_11_server  |
z21_11_server  | Received packet # 4, length: 512, content:
z21_11_server  | =>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[
z21_11_server    |  Sending  confirmation  for  packet  #  4
z21_11_server  | z21_11_client  | Sent packet 5 and received
confirmation z21_11_server  | z21_11_server  | Received packet
# 5, length: 512, content:
z21_11_server  | \]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

```
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
z21_11_server    |  Sending  confirmation  for  packet  #  5
z21_11_server  | z21_11_server  | z21_11_server  | Received
packet  #  6,  length:  512,  content:  z21_11_server   |  {|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:   z21_11_server      |   Sending
confirmation for packet # 6 z21_11_server  | z21_11_client
| Sent packet 6 and received confirmation z21_11_client  |
Sent  packet  7  and  received  confirmation  z21_11_server   |
z21_11_server  | Received packet # 7, length: 512, content:
z21_11_server  | ;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY
z21_11_server     |   Sending   confirmation   for   packet   #   7
z21_11_server   |  z21_11_server   |  z21_11_server   |  Received
packet # 8, length: 512, content:
z21_11_server  | Z[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx
z21_11_server    |  Sending  confirmation  for  packet  #  8
z21_11_server  | z21_11_client  | Sent packet 8 and received
confirmation  z21_11_server   |  z21_11_server   |  Received
packet # 9, length: 512, content:
```

```
z21_11_server  | yz{|}~ !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./012345678   z21_11_server      |   Sending
confirmation for packet # 9 z21_11_server  | z21_11_client
| Sent packet 9 and received confirmation z21_11_client
exited with code 0
```

Uwagi dot. problemów

Największym wyzwaniem było zapewnienie poprawnej komunikacji między klientem a serwerem, szczególnie przy weryfikacji poprawności otrzymanych danych. Problem ten rozwiązano poprzez dodanie szczegółowej weryfikacji danych w serwerze. Istotne okazało się także odwracanie kolejności bajtów w „nagłówku”, za pomocą funkcji htons i htonl – ze względu na konieczność konwersji ze standardu hosta na standard sieciowy (używany do transmisji danych). Względem wersji UDP należało oczywiście dostosować kod do komunikacji poprzez TCP.