*27.11.2023*

Wersja 1.0.

**Programowanie Sieci Internetowych**

**Zespół: [Nazwa Zespołu]**

**Autorzy:**

**[…]**

**Treść Zadania**

Zadanie polega na stworzeniu aplikacji klient-serwer komunikującej się za pomocą protokołu UDP. Klient (napisany w języku C) wysyła datagramy o ustalonym rozmiarze zawierające kolejne drukowalne znaki ASCII. Serwer (napisany w Pythonie) odbiera datagramy, weryfikuje poprawność danych i odsyła potwierdzenie odbioru.

**Opis Rozwiązania Problemu**

Klient (C)

/\* Create socket. \*/

sock = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sock == -1) bailout("Nie można utworzyć gniazda");

/\* Configure server. \*/

/\* uzyskajmy adres IP z nazwy . \*/

server.sin\_family = AF\_INET;

hp = gethostbyname2(argv[1], AF\_INET);

/\* hostbyname zwraca strukture zawierajaca adres danego hosta \*/

if (hp == (struct hostent \*) 0) errx(2, "%s: unknown host\n", argv[1]);

printf("address resolved...\n");

memcpy((char \*) &server.sin\_addr, (char \*) hp->h\_addr, hp->h\_length);

server.sin\_port = htons(atoi(argv[2]));

connect (sock, (struct sockaddr \*) &server, sizeof(server) );

for (packet\_number = 0; packet\_number < atoi(argv[3]); ++packet\_number)

{

usleep(udelay);

int32\_t packet\_number\_net = htonl(packet\_number);

int16\_t data\_length\_net = htons(DGRAMSIZE - 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 < DGRAMSIZE; i++)

{

buffer[i] = current\_char;

current\_char++;

if (current\_char > ASCII\_END) current\_char = ASCII\_START;

}

//Wyslanie datagramu

if (sendto(sock, buffer, DGRAMSIZE, 0, (struct sockaddr \*)&server, sizeof(server)) < 0)

{

perror("Nie można wysłać datagramu\n");

continue;

}

// Odbieranie odpowiedzi

if (recvfrom(sock, recv\_buffer, DGRAMSIZE, 0, (struct sockaddr \*)&server, &addr\_len) < 0)

{

perror("Błąd przy odbieraniu odpowiedzi\n");

continue;

}

// Weryfikacja odpowiedzi

if (strncmp(recv\_buffer, "OK", 2) != 0)

{

perror("Otrzymano niepoprawną odpowiedź od serwera\n");

continue;

}

//Wszystko poszlo OK

printf("Wysłano pakiet nr %d i otrzymano potwierdzenie\n", packet\_number);

}

close(sock);

Klient tworzy gniazdo i konfiguruje połączenie z serwerem. W głównej pętli tworzy datagramy UDP 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 *sendto*. Następnie czeka na potwierdzenie weryfikacji od serwera (*recvfrom*). Weryfikuje potwierdzenie, aby sprawdzić czy wszystko jest ok. (porównanie napisów za pomocą *strncmp*). Następnie inkrementuje numer następnego pakietu, zasypia na chwilę i kontynuuje działanie od nowa.

Serwer (Python)

with socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) as sock:

sock.bind((HOST, PORT))

packet\_number = -1

while True:

data, addr = sock.recvfrom(BUFSIZE)

if not data:

print("Error in datagram?")

break

expected\_packet\_number = packet\_number + 1

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("Missing packet! Expected {}, received {}".format(expected\_packet\_number, packet\_number))

sock.sendto(b'Missing packet', addr)

continue

if received\_length != data\_length + 6:

print("Incorrect data length. Expected {}, received {}".format(data\_length + 6, received\_length))

sock.sendto(b'Incorrect data length', addr)

continue

if not is\_printable\_ascii(message\_data):

print("Data is not printable ASCII")

sock.sendto(b'Data is not printable ASCII', addr)

continue

print("\nReceived packet # {}, length: {}, content: \n{}".format(packet\_number, received\_length, message\_data.decode('ascii', errors='ignore')))

sock.sendto(b'OK', addr)

print('Sending confirmation for packet #', packet\_number)

Serwer tworzy gniazdo i nasłuchuje na określonym porcie. Odbiera datagramy (sock.recvfrom) 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*). Serwer odsyła potwierdzenie odbioru do klienta po każdym otrzymanym datagramie. (*sendto*)

Po każdym odebranym pakiecie, inkrementuje oczekiwany numer pakietu.

**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).

**Opis Konfiguracji Testowej**

* **Adres IP Serwera:** 127.0.0.1 (interfejs: **lo** (host lokalny))
* **Port Serwera:** 12345
* **Rozmiar Datagramu:** 512 bajtów

**Opis Testowania i Wydruki z Testów**

Testowanie polegało na uruchomieniu serwera i klienta, a następnie obserwowaniu logów serwera podczas odbierania i przetwarzania datagramów. [*Serwer poprawnie identyfikował nieprawidłowości w danych i utraty pakietów – to będzie można wspomnieć w Z 1.3*].

**Wnioski Końcowe i Uwagi Własne**

Projekt pokazał praktyczne zastosowanie protokołów UDP w komunikacji sieciowej. Zostały poruszone kluczowe aspekty programowania sieciowego, takie jak obsługa datagramów, weryfikacja danych i obsługa błędów.