

Politechnika Warszawska

PSIR: Laboratorium 1

Podstawy tworzenia i uruchamiania oprogramowania sieciowego
bazującego na gniazdach w systemie Linux

Kazimierz Kochan 303704

Monika Lewandowska 303707

2022-03-16

Zadanie 1.....	2
Zadanie 2.....	4

Wstęp

Laboratorium składa się z dwóch zadań pozwalających w sposób praktyczny przyswoić wiedzę z wykładu dotyczącą tworzenia i uruchamiania oprogramowania sieciowego w systemie Linux bazującego na socketach. Oba ćwiczenia laboratoryjne wykonaliśmy w parze stosując się poleceń z instrukcji – Kochan_Kazimierz_Cyryl.pdf. Plik znajdują się w repozytorium GIT Kazimierz Kochana w folderze lab1 oraz odpowiednich folderach zad1 i zad2.

Zadanie 1

Zakładając architekturę klient-serwer, napisać w języku C: kod klienta i kod serwera. Tak aby klient cyklicznie przysyłał wiadomości będące tekstową reprezentacją aktualnego czasu. Komunikacja między klientem a serwerem ma być realizowana na porcie serwera o numerze: 3792 protokołu TCP. Klient niech wysyła cyklicznie wiadomości co 1030 ms. Dodatkowo serwer po odebraniu każdej wiadomości niech wypisuje ją na swojej konsoli.

Udało się zaimplementować program zapewniający komunikację między klientem a serwerem wykorzystując mechanizm gniazd (sockets).

- Adres IP klienta: 192.168.56.111
- Adres IP serwera: 192.168.56.110

Skomentowany kod programu klienta i serwera w języku C znajduje się w repozytorium Git razem z raportem.

Obydwa programy mogą przyjąć 0 lub 1 argument. W przypadku zerowej liczby argumentów programy działają w nieskończonej pętli. Jako 1 argument można podać „-h”/”-help”, które skutkuje wyświetleniem informacji o składni, lub liczbę sekund, przez jaką program ma pracować.

Program serwera jest odpowiedzialny za utworzenie gniazda i powiązanie go z portem lokalnej maszyny. Po uruchomieniu wypisywana jest na konsoli informacja “PSIR 22L Lab1, exercise 1: Simple TCP server” o starcie serwera. Następnie serwer nasłuchuje na porcie 3792 na klienta (w pętli) i gdy ten się połączy serwer wypisuje komunikat o połączeniu klienta i jego adresie IP oraz rozpoczyna wypisywać otrzymane od klienta dane (tutaj jest to data wysłania wiadomości przez klienta). Gdy klient zakończy komunikację (tutaj sygnał przerwania lub koniec czasu podanego jako argument przy uruchamianiu programu klienta) serwer informuje o odłączeniu klienta wypisując komunikat “Client has disconnected”.

Na poniższym zrzucie ekranu widać wynik działania programów serwera i klienta.

```
Overview x 192.168.56.110 (Ad Hoc) x 192.168.56.110 (Ad Hoc) x
psir@psir21z:~$ ./server
PSIR 22L Lab1, exercise 1: Simple TCP server
Infinite loop
Listening for the client
Client with IP: 192.168.56.111 has connected. New sock desc.: 4
Client sent data: Fri Apr 8 20:34:31 2022
Client sent data: Fri Apr 8 20:34:32 2022
Client sent data: Fri Apr 8 20:34:33 2022
Client sent data: Fri Apr 8 20:34:34 2022
Client sent data: Fri Apr 8 20:34:35 2022
Client sent data: Fri Apr 8 20:34:36 2022
Client sent data: Fri Apr 8 20:34:37 2022
Client sent data: Fri Apr 8 20:34:38 2022
Client sent data: Fri Apr 8 20:34:39 2022
Client sent data: Fri Apr 8 20:34:40 2022
Client sent data: Fri Apr 8 20:34:41 2022
Client sent data: Fri Apr 8 20:34:42 2022
Client sent data: Fri Apr 8 20:34:43 2022
Client sent data: Fri Apr 8 20:34:44 2022
Client sent data: Fri Apr 8 20:34:45 2022
Client sent data: Fri Apr 8 20:34:46 2022
Client sent data: Fri Apr 8 20:34:47 2022
Client sent data: Fri Apr 8 20:34:48 2022
Client sent data: Fri Apr 8 20:34:49 2022
Client sent data: Fri Apr 8 20:34:50 2022
Client sent data: Fri Apr 8 20:34:51 2022
Client sent data: Fri Apr 8 20:34:53 2022
Client sent data: Fri Apr 8 20:34:54 2022
Client sent data: Fri Apr 8 20:34:55 2022
Client sent data: Fri Apr 8 20:34:56 2022
Client sent data: Fri Apr 8 20:34:57 2022
Client sent data: Fri Apr 8 20:34:58 2022
Client sent data: Fri Apr 8 20:35:00 2022
Client sent data: Fri Apr 8 20:35:01 2022
Client has disconnected
Server has worked for 38.0 seconds
Quitting...

Overview x 192.168.56.11... x 192.168.56.11... x 192.168.56.111...
psir@psir21z:~$ ./client 30
PSIR 22L Lab1, exercise 1: Simple TCP client
Planned uptime: 30.0 seconds
Sending current time: Fri Apr 8 20:34:31 2022
Sending current time: Fri Apr 8 20:34:32 2022
Sending current time: Fri Apr 8 20:34:33 2022
Sending current time: Fri Apr 8 20:34:34 2022
Sending current time: Fri Apr 8 20:34:35 2022
Sending current time: Fri Apr 8 20:34:36 2022
Sending current time: Fri Apr 8 20:34:37 2022
Sending current time: Fri Apr 8 20:34:38 2022
Sending current time: Fri Apr 8 20:34:39 2022
Sending current time: Fri Apr 8 20:34:40 2022
Sending current time: Fri Apr 8 20:34:41 2022
Sending current time: Fri Apr 8 20:34:42 2022
Sending current time: Fri Apr 8 20:34:43 2022
Sending current time: Fri Apr 8 20:34:44 2022
Sending current time: Fri Apr 8 20:34:45 2022
Sending current time: Fri Apr 8 20:34:46 2022
Sending current time: Fri Apr 8 20:34:47 2022
Sending current time: Fri Apr 8 20:34:48 2022
Sending current time: Fri Apr 8 20:34:49 2022
Sending current time: Fri Apr 8 20:34:50 2022
Sending current time: Fri Apr 8 20:34:51 2022
Sending current time: Fri Apr 8 20:34:53 2022
Sending current time: Fri Apr 8 20:34:54 2022
Sending current time: Fri Apr 8 20:34:55 2022
Sending current time: Fri Apr 8 20:34:56 2022
Sending current time: Fri Apr 8 20:34:57 2022
Sending current time: Fri Apr 8 20:34:58 2022
Sending current time: Fri Apr 8 20:34:59 2022
Sending current time: Fri Apr 8 20:35:00 2022
Sending current time: Fri Apr 8 20:35:01 2022
Client has worked for 30.0 seconds
Quitting...
```

Rys. 1 Komunikacja między serwerem a klientem

Wykorzystaliśmy też polecane nam na zajęciach narzędzie do zapisu ruchu sieciowego do pliku "tshark" zbierając pakiety na odpowiednim interfejsie (poza pakietami na porcie 22 przypisanym do usługi ssh).

```
root@psir21z:/home/psir# /usr/bin/tshark -n -i enp0s8 -w pakiety.pcap "
not tcp port 22"
Running as user "root" and group "root". This could be dangerous.
Capturing on 'enp0s8'
75 ^C
```

Rys. 2 Zapis ruchu sieciowego do pliku

Następnie dokonaliśmy analizy zebranego ruchu sieciowego w programie Wireshark.

- Jak widać na poniższym zrzucie ekranu zestawiona została komunikacja między serwerem (192.168.56.110), a klientem (192.168.56.111) z wykorzystaniem protokołu TCP na porcie 3792.
- Wiadomości są wysyłane przez klienta zgodnie z poleceniem co ok. 1030ms.

No.	Time	Source	Destination	Protocol	Length	Info
2	12.558543349	192.168.56.111	192.168.56.110	TCP	74	53514 → 3792 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1999872218 TSecr=0
3	12.558575620	192.168.56.110	192.168.56.111	TCP	74	3792 → 53514 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1999871188 TSecr=1233773861
4	12.558830413	192.168.56.111	192.168.56.110	TCP	66	53514 → 3792 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1999871188 TSecr=1233773861
5	12.559544320	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=25 TSval=1999871188 TSecr=1233773861
6	12.559557080	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=26 Win=65152 Len=0 TSval=1233773861 TSecr=1
7	13.589691844	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=26 Ack=1 Win=64256 Len=25 TSval=1999872218 TSecr=1
8	13.589818520	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=51 Win=65152 Len=0 TSval=1233774891 TSecr=1
9	14.620394166	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=51 Ack=1 Win=64256 Len=25 TSval=1999873249 TSecr=1
10	14.620427585	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=76 Win=65152 Len=0 TSval=1233775922 TSecr=1
11	15.651290192	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=76 Ack=1 Win=64256 Len=25 TSval=1999874279 TSecr=1
12	15.651316906	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=101 Win=65152 Len=0 TSval=1233776953 TSecr=1
13	16.681206438	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=101 Ack=1 Win=64256 Len=25 TSval=1999875310 TSecr=1
14	16.681235929	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=126 Win=65152 Len=0 TSval=1233777983 TSecr=1
17	17.711936073	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=126 Ack=1 Win=64256 Len=25 TSval=1999876340 TSecr=1
18	17.712038879	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=151 Win=65152 Len=0 TSval=1233779014 TSecr=1
19	18.742579523	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=151 Ack=1 Win=64256 Len=25 TSval=1999877371 TSecr=1
20	18.742651392	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=176 Win=65152 Len=0 TSval=1233780044 TSecr=1
21	19.773516163	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=176 Ack=1 Win=64256 Len=25 TSval=1999878402 TSecr=1
22	19.773637940	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=201 Win=65152 Len=0 TSval=1233781075 TSecr=1
23	20.804133110	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=201 Ack=1 Win=64256 Len=25 TSval=1999879433 TSecr=1
24	20.804158335	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=226 Win=65152 Len=0 TSval=1233782106 TSecr=1
25	21.834521060	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=226 Ack=1 Win=64256 Len=25 TSval=1999880463 TSecr=1
26	21.834546913	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=251 Win=65152 Len=0 TSval=1233783136 TSecr=1
27	22.864788798	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=251 Ack=1 Win=64256 Len=25 TSval=1999881493 TSecr=1
28	22.864811981	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=276 Win=65152 Len=0 TSval=1233784166 TSecr=1
29	23.895331304	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=276 Ack=1 Win=64256 Len=25 TSval=1999882524 TSecr=1
30	23.895355816	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=301 Win=65152 Len=0 TSval=1233785137 TSecr=1
31	24.925626522	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK] Seq=301 Ack=1 Win=64256 Len=25 TSval=1999883554 TSecr=1
32	24.925651890	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=1 Ack=326 Win=65152 Len=0 TSval=1233786227 TSecr=1

Rys. 3 Analiza przesyłanych pakietów w narzędziu Wireshark

- Poniższe zrzuty ekranu przedstawiają zawarte w pakietach dane z datą dwóch pierwszych przesyłanych wiadomości, które zgadzają się z tymi wypisywanymi na konsoli przez program.

5	12.559544320	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK]
6	12.559557080	192.168.56.110	192.168.56.111	TCP	66	3792 → 53514 [ACK] Seq=
> Frame 5: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface enp0s8, id 0 > Ethernet II, Src: PcsCompu_d6:20:0c (08:00:27:d6:20:0c), Dst: PcsCompu_c4:30:0b (08:00:27:c4:30:0b) > Internet Protocol Version 4, Src: 192.168.56.111, Dst: 192.168.56.110 > Transmission Control Protocol, Src Port: 53514, Dst Port: 3792, Seq: 1, Ack: 1, Len: 25 > Data (25 bytes) Data: 467269204170722020382032303a33343a333120323032320a						
0000	08 00 27 c4 30 0b 08 00	27 d6 20 0c 08 00 45 00	..0... ..E.			
0010	00 4d 85 46 40 00 40 06	c3 36 c0 a8 38 6f c0 a8	.M.G@. .5..8o..			
0020	38 6e d1 0a 0e d0 07 15	c5 c0 5b eb 96 60 80 18	8n.....[... ..			
0030	01 f6 c0 2c 00 00 01 01	08 0a 77 33 a0 da 49 89w3..I.			
0040	e5 24 46 72 69 20 41 70	72 20 20 38 20 32 30 3a	..%Fri Ap r 8 20:			
0050	33 34 3a 33 31 20 32 30	32 32 0a	34:31 20 22.			

Rys. 4 Pierwszy pakiet wysłany przez klienta z datą

7	13.589691844	192.168.56.111	192.168.56.110	TCP	91	53514 → 3792 [PSH, ACK]
> Frame 7: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface enp0s8, id 0 > Ethernet II, Src: PcsCompu_d6:20:0c (08:00:27:d6:20:0c), Dst: PcsCompu_c4:30:0b (08:00:27:c4:30:0b) > Internet Protocol Version 4, Src: 192.168.56.111, Dst: 192.168.56.110 > Transmission Control Protocol, Src Port: 53514, Dst Port: 3792, Seq: 26, Ack: 1, Len: 25 > Data (25 bytes) Data: 467269204170722020382032303a33343a333220323032320a						
0000	08 00 27 c4 30 0b 08 00	27 d6 20 0c 08 00 45 00	..0... ..E.			
0010	00 4d 85 47 40 00 40 06	c3 35 c0 a8 38 6f c0 a8	.M.G@. .5..8o..			
0020	38 6e d1 0a 0e d0 07 15	c5 c0 5b eb 96 60 80 18	8n.....[... ..			
0030	01 f6 bb 0c 00 00 01 01	08 0a 77 33 a0 da 49 89w3..I.			
0040	e5 25 46 72 69 20 41 70	72 20 20 38 20 32 30 3a	..%Fri Ap r 8 20:			
0050	33 34 3a 33 32 20 32 30	32 32 0a	34:32 20 22.			

Rys. 4 Drugi pakiet wysłany przez klienta z datą

Czas między pierwszą, a drugą wiadomością: 13.58969 - 12.55954 = 1.03015[s] czyli ok. 1030ms

Zadanie 2

Zakładając architekturę w której mamy wielu uruchomionych jednocześnie klientów oraz jeden działający serwer, napisać w języku C kod serwera i kod klientów (wszystkie mogą mieć identyczną implementację, ale podczas ich działania serwer musi je rozróżniać w zaproponowany przez implementatora sposób). Po stracie serwer powinien w nieskończoność nasłuchiwać na porcie 19398 protokołu UDP, zgłoszeń o pojawieniu się każdego nowego klienta (wymagane jest przesłanie tzw. wiadomości HELLO - samodzielnie zaproponuj format tej wiadomości). Równolegle do tego zadania ma też co losowy czas wachający się między 100 a 2310 ms, wysłać losowo wybranemu klientowi (z zestawu aktualnie znanych) wiadomość o losowej treści, budowanej z dokładnie 18 znaków (każdy znak może być dowolnym znakiem: A..Za..z0..9). Każdy klient po odebraniu takiej wiadomości ma odesłać ją serwerowi, dodatkowo dopisując na jej początku: "Re:". Natomiast serwer po odebraniu wiadomości od każdego z klientów, w swojej konsoli prezentuje takie wiadomości dodając na jej początku datę i czas otrzymania pakietu UDP od zgłoszonych klientów. Pamiętaj, że elementy systemu muszą rozróżniać wiadomości: HELLO i te o losowej treści - wskazane jest zatem utworzenie dodatkowego nagłówka wiadomości warstwy aplikacji.

Kody źródłowe serwera i klienta znajdują się na repozytorium GIT. Posiadają one informatywne komentarze. Kod jest długi i szeroki, przez co, czytanie go w sprawozdaniu nie byłoby wygodne. Wszystkie założenia udało nam się zaimplementować.

Przyjęte założenia do implementacji:

- 18 znaków - długość wiadomości przesyłanej przez serwer
- dodatkowy nagłówek wiadomości warstwy aplikacji. – 1 bajt, maks 127 treść (128 all)
- 'm' - wiadomość zwykła
- 'h' - wiadomość HELLO (rejestracja nowego klienta)
- 'g' – wiadomość GOODBYE (chęć odłączenia klienta)

Informacje na temat rozwiązania

Aby możliwe było równoległe nasłuchiwanie na nowych klientów oraz wysyłanie do nich wiadomości nasze rozwiązanie wykorzystuje technikę zwaną „self-pipe trick”. Stworzyliśmy prywatny dla procesu pipe oraz dodatkowy wątek. Wątek losuje czas, przez który śpi. Gdy się obudzi wysyła przez pipe wiadomość, której odebranie na drugim końcu pipe’a sygnalizowane jest przez metodę select. Natępnie takie zdarzenie, analogicznie jak dla wiadomości odebranych przez socket, jest odpowiednio procesowane. Błędy wypisywane są na standardowe wyjście błędów. Klienci przechowywani są w tablicy, o określonej maksymalnej wielkości. Dane, z założeń, które zostały dla nas wygenerowane można samodzielnie ustawić za pomocą makr na początku programu. Dodaliśmy dodatkową możliwość ustalenia czasu pracy, aby ułatwić obsługę programów oraz nie wymuszać za każdym razem ręcznego zamknięcia procesu.

Demonstracja

Programy można uruchomić z flagami wyświetlającymi pomoc. W przypadku podania niepoprawnej liczby argumentów program nie zostaje uruchomiony.

```
psir@psir21z:~/psir22L/lab1/zad2$ ./server -help
Syntax: ./server [uptime (0=infinity)]
psir@psir21z:~/psir22L/lab1/zad2$ ./server -h
Syntax: ./server [uptime (0=infinity)]
psir@psir21z:~/psir22L/lab1/zad2$ ./server 1241 34 21342
Syntax: ./server [uptime (0=infinity)]
psir@psir21z:~/psir22L/lab1/zad2$
```

Rysunek 1 Obsługa argumentów

W scenariuszu testowym uruchomiliśmy serwer na maszynie 192.168.56.110, oraz 4 klientów: 2 na tej samej maszynie oraz 2 na hoście 192.168.56.111. Klienci mieli ustawione różne czasy pracy: 1000, ∞, 18 oraz 35 sekund.

```

psir@psir21z:~/psir22L/lab1/zad2$ ./server
PSIR 22L Lab1, exercise 2: Simple UDP server
Infinite loop
23:19:58: Starting thread
Sleeping for: 1707 ms
23:19:58: Received HELLO message from (192.168.56.110:36705): HELLO!
Added a new client
23:19:59: Received HELLO message from (192.168.56.110:37253): HELLO!
Added a new client
Sleeping for: 1704 ms
23:19:59: Sent dgram: 231959wJ1aPEh9UBZE to registered client (192.168.56.110:36705)
23:19:59: Received message from registered client(192.168.56.110:36705): Re:231959wJ1aPEh9UBZE
23:20:00: Received HELLO message from (192.168.56.111:39458): Hello!
Added a new client
Sleeping for: 1700 ms
23:20:01: Sent dgram: 232001KdjZVMocSUEi to registered client (192.168.56.111:39458)
23:20:01: Received message from registered client(192.168.56.111:39458): Re:232001KdjZVMocSUEi
Sleeping for: 1372 ms
23:20:03: Sent dgram: 232003pjaKQ2IwwPvh to registered client (192.168.56.110:37253)
23:20:03: Received message from registered client(192.168.56.110:37253): Re:232003pjaKQ2IwwPvh
23:20:03: Received HELLO message from (192.168.56.111:42871): Hello!
Added a new client
Sleeping for: 1964 ms
23:20:04: Sent dgram: 232004JH1jJa8uKkwt to registered client (192.168.56.110:37253)
23:20:04: Received message from registered client(192.168.56.110:37253): Re:232004JH1jJa8uKkwt
Sleeping for: 1857 ms
23:20:05: Sent dgram: 232005Zg... to registered client (192.168.56.111:39458)

```

Rysunek 2 Działanie serwera UDP

Jak widać na powyższym zrzucie ekranu serwer odebrał wiadomości HELLO od 4 klientów, zarejestrował ich oraz zaczął wysyłać wiadomości do losowych klientów. Wiadomości wysyłane są co losowy czas, dobrany według polecenia (miedzy 100 a 2310 ms). Klienci są rozróżniani pomiędzy sobą przez adres Ipv4 oraz port. Wiadomość składa się z 18 znaków z podanej dziedziny. Pierwsze 6 to godzina, minuty i sekundy, co ułatwia obserwowanie działania. Wiadomości, które serwer odbiera od klientów drukowane są na standardowe wyjście.

```

psir@psir21z:~/psir22L/lab1/zad2$ ./client 18
PSIR 2022L Lab1, exercise 2: Simple UDP client
Planned uptime: 18.0 seconds
Sending HELLO message
Waiting for server messages...
Received message: "232003pjaKQ2IwwPvh"
Sending message: "mRe:232003pjaKQ2IwwPvh"
Received message: "232004JH1jJa8uKkwt"
Sending message: "mRe:232004JH1jJa8uKkwt"
Received message: "232011N0m2agEijUvz"
Sending message: "mRe:232011N0m2agEijUvz"
Quitting...
Sending GOODBYE message
Client has worked for 18.0 seconds
psir@psir21z:~/psir22L/lab1/zad2$ █

```

Rysunek 3 Klient działający określony czas

Klient wysyła wiadomość HELLO do serwera. Następnie do wiadomości, które otrzyma dodaje prefiks Re: i odsyła je do serwera. Po upływie czasu pracy określonego jako argument wysyła wiadomość GOODBYE i kończy swoje działanie.


```

psir@psir21z:~/psir22L/lab1/zad2$ ./client
PSIR 2022L Lab1, exercise 2: Simple UDP client
Infinite loop
Sending HELLO message
Waiting for server messages...
Received message: "232001KdjZVMocSUEi"
Sending message: "mRe:232001KdjZVMocSUEi"
Received message: "232006z7m0enmVdUfz"
Sending message: "mRe:232006z7m0enmVdUfz"
Received message: "232013Qvpx20GXISL2"
Sending message: "mRe:232013Qvpx20GXISL2"
Received message: "232018M09ycf0VzLuw"
Sending message: "mRe:232018M09ycf0VzLuw"

```

Rysunek 4 Wieczny klient

Wieczny klient, działa tak długo, aż zostanie siłą zamknięty przez użytkownika, zamknięty przez psujący się z upływu czasu komputer lub wystąpienie błędu (na przykład z komunikacją sieciową z serwerem).

```

Sleeping for: 1877 ms
23:22:44: Sent dgram: 232244SWHAcLffnSAp to registered client (192.168.56.110:44164)
23:22:44: Received message from registered client(192.168.56.110:44164): Re:232244SWHAcLffnSAp
23:22:45: Received GOODBYE message from registered client(192.168.56.110:44164): GOODBYE!
23:22:45: Client (192.168.56.110:44164) has quit
Sleeping for: 556 ms
Sleeping for: 415 ms
Sleeping for: 192 ms
Sleeping for: 2172 ms
Sleeping for: 1813 ms

```

Rysunek 5 Serwer bez klientów

Gdy serwer nie ma zapisanego w bazie żadnego klienta nasłuchuje on w nieskończoność, aż dołączą oni, wysyłając wiadomość HELLO.

```

psir@psir21z:~/psir22L/lab1/zad2$ ./server 10
PSIR 22L Lab1, exercise 2: Simple UDP server
Planned uptime: 10.0 seconds
23:31:42: Starting thread
Sleeping for: 2209 ms
23:31:43: Received HELLO message from (192.168.56.110:46286): HELLO!
Added a new client
Sleeping for: 2234 ms
23:31:45: Sent dgram: 233145eLTtNxEcvfu7 to registered client (192.168.56.110:46286)
23:31:45: Received message from registered client(192.168.56.110:46286): Re:233145eLTtNxEcvfu7
Sleeping for: 1672 ms
23:31:47: Sent dgram: 233147UAMY3WQMAGLL to registered client (192.168.56.110:46286)
23:31:47: Received message from registered client(192.168.56.110:46286): Re:233147UAMY3WQMAGLL
Sleeping for: 197 ms
23:31:48: Sent dgram: 233148WoQYxAfaXHDh to registered client (192.168.56.110:46286)
23:31:48: Received message from registered client(192.168.56.110:46286): Re:233148WoQYxAfaXHDh
Sleeping for: 846 ms
23:31:49: Sent dgram: 233149cPaWeKJ7upJU to registered client (192.168.56.110:46286)
23:31:49: Received message from registered client(192.168.56.110:46286): Re:233149cPaWeKJ7upJU
Sleeping for: 2067 ms
23:31:50: Sent dgram: 233150w0Wg2D2p37o0 to registered client (192.168.56.110:46286)
23:31:50: Received message from registered client(192.168.56.110:46286): Re:233150w0Wg2D2p37o0
Quitting...
Server has worked for 10.0 seconds
psir@psir21z:~/psir22L/lab1/zad2$ █

```

Rysunek 6 Serwer z określonym czasem pracy

Możliwe jest określenie planowanego czasu pracy serwera. Po jego upływie serwer kończy swoje działanie.

Wnioski

Udało nam się zrealizować obydwa zadania, czego wyniki udokumentowaliśmy na powyższych zrzutach ekranu, a kod serwera i klienta dla obydwu zadań umieściliśmy w dedykowanym repozytorium Git.

Samodzielna implementacja architektury klient-serwer (oraz w drugim zadaniu kilku klientów-serwer) pozwoliła nam lepiej zrozumieć wiedzę przekazaną na wykładzie i przetestować oprogramowanie sieciowe w systemie Linux bazujące na socketach.