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

Wstęp

Laboratorium składa się z dwóch zadan 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 przesyłał wiadomości będące tekstową reprezenacją 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 (sockety).

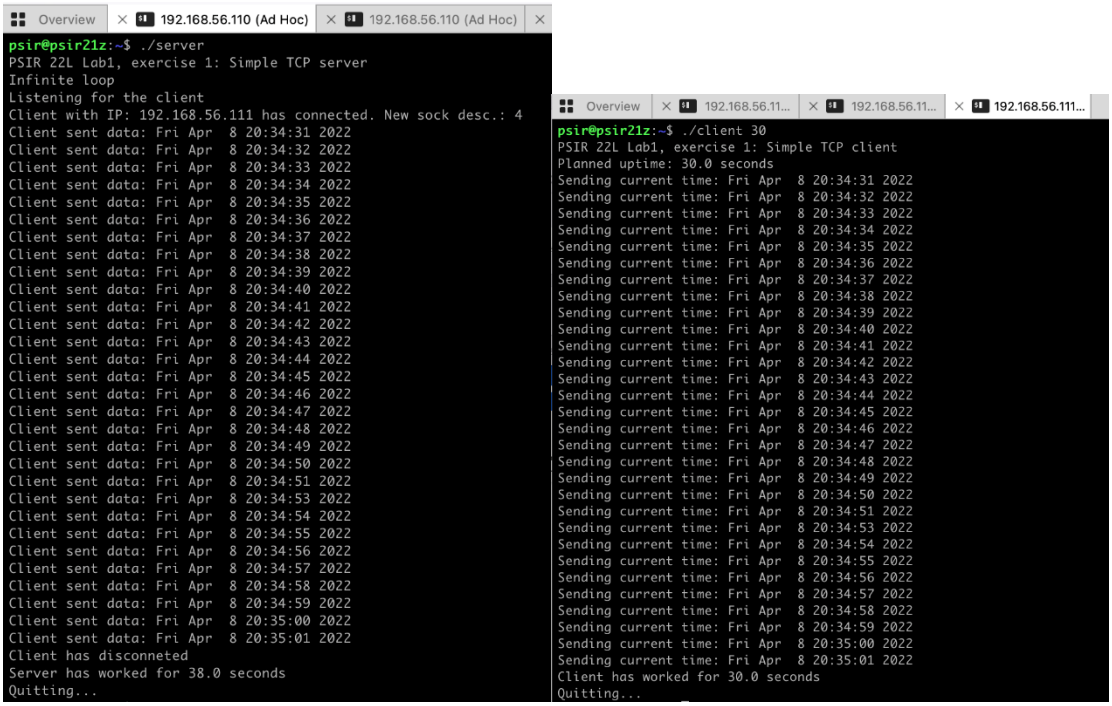
- Adres IP klienta: 192.168.56.111
- Adres IP serwera: 192.168.56.110

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.

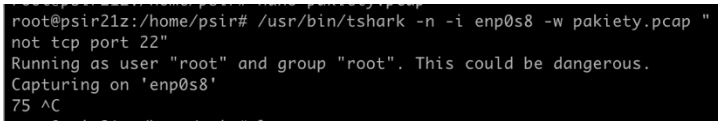
Program serwera jest odpowiedzialny za utworzenie gniazda i powiązanie go z portem lokalnej maszyny. Po uruchomieniu wypisywana jest na konsoli informacja “PSIR 22L Lab1, exercise 1: Simple TCP server” o starcie serwera. Następnie serwer nasłuchuje na porcie 3792 na klienta (w pętli) i gdy ten się połączy serwer wypisuje komunikat o połaczeniu klienta i jego adresie IP oraz rozpoczyna wypisywać otrzymane od klienta dane (tutaj jest to data wysłania wiadomości przez klienta). Gdy klient zakonczy 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.



Rys. 1 Komunikacja między serwerem a klientem

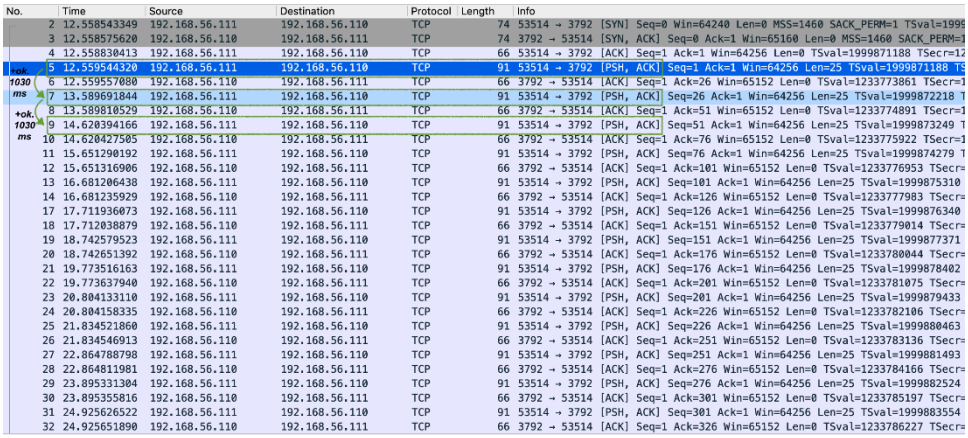
Wykorzystaliśmy też polecone nam na zajęciach narzędzie do zapisu ruchu sieciowego do pliku “tshark” zbierając pakiety na odpowiednim interfejsie (poza pakietami na porcie 22 przypisanym do usługi ssh).



Rys. 2 Zapis ruchu sieciowego do pliku

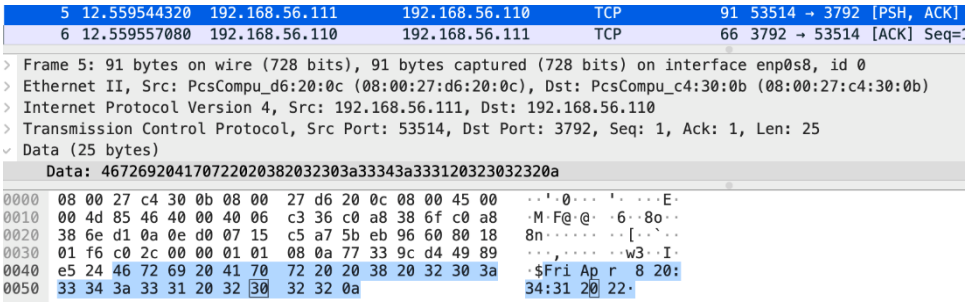
Następnie dokonaliśmy analizy zebranego ruchu sieciowego w programie Wireshark.

- 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.

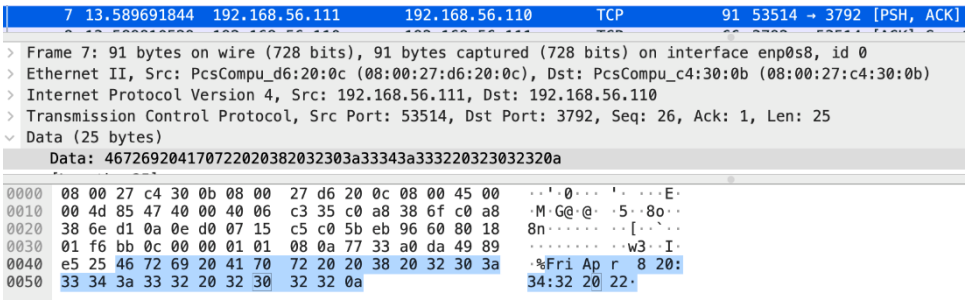


Rys. 3 Analiza przesyłanych pakietów w narzędziu Wireshark

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



Rys. 4 Pierwszy pakiet wysłany przez klienta z datą



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 wachajacy się miedzy 100 a 2310 ms, wysyłać 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 wiadomści nasze rozwiązanie wykorzystuje techikę 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, analigocznie 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.

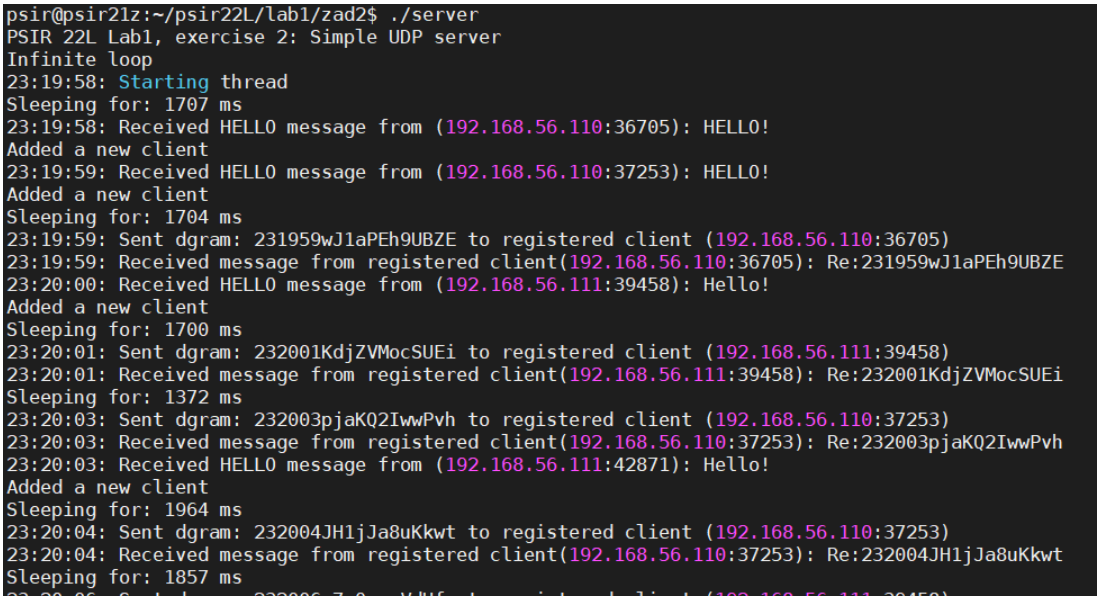
Demonstracja

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$
```

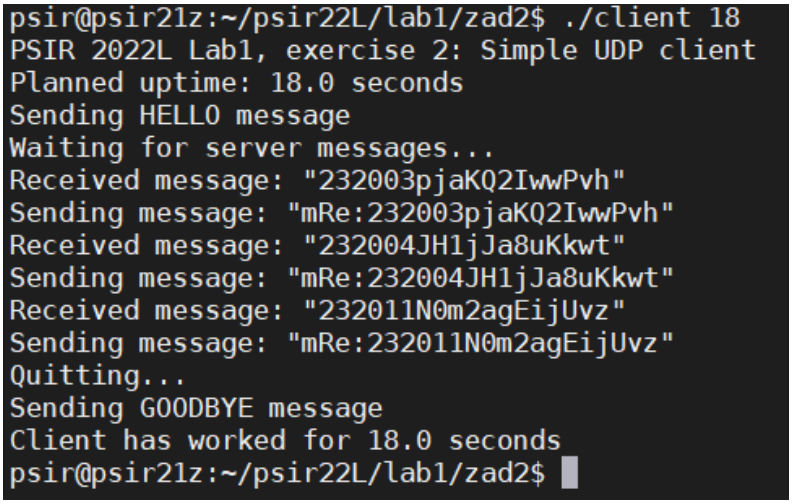
Rysunek 1 Obsługa argumentów

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.



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.



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: "232006z7mOenmVdUfz"
Sending message: "mRe:232006z7mOenmVdUfz"
Received message: "232013Qvpx2OGXISL2"
Sending message: "mRe:232013Qvpx2OGXISL2"
Received message: "232018MO9ycf0VzLuw"
Sending message: "mRe:232018MO9ycf0VzLuw"
```

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: 233150wOWg2D2p37oO to registered client (192.168.56.110:46286)
23:31:50: Received message from registered client(192.168.56.110:46286): Re:233150wOWg2D2p37oO
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 upłynięciu 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 zadan 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.