programowanie usług sieciowych

„Protokół UDP”

laboratorium: 02

Spis treści

Spis treści 2
1. Wiadomości wstępne 3
1.1. Tematyka laboratorium 3
1.2. Zagadnienia do przygotowania 3
2. Przebieg laboratorium 4
2.1. Przygotowanie laboratorium 4
2.2. Zadanie 1. Komunikacja klient – serwer 4
2.3. Zadanie 2. Klient i serwer echa za pomocą protokołu UDP 5
2.4. Zadanie 3. Modyfikacja poprzednich programów 6
2.5. Zadanie 4. Kojarzenie adresu zdalnego z gniazdem 6
2.6. Zadanie 5. Brak sterowania przepływem w protokole UDP 7
3. Opracowanie i sprawozdanie 9

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące warstwy transportowej modelu ISO/OSI a w szczególności **protokołu UDP**. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji klient-serwer w oparciu o **protokół UDP** oraz badanie jego właściwości. Protokół UDP (ang. *User Datagram Protocol*) znajduje się w warstwie transportowej modelu ISO/OSI. Korzysta z protokołu IPv4 lub IPv6 tworząc gniazda.

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi **protokołu UDP**:

- specyfikacja UDP; [RFC 768]
- budowa nagłówka UDP; [RFC 768]

A także z zagadnieniami dotyczącymi **gniazd UDP**:

- funkcje gniazd UDP; [Stevens, 248]

Programy:

- ethereal;
- netstat.

Literatura:

[1] W.R. Stevens, *„Programowanie Usług Sieciowych”*, *„API: gniazda i XTI”*.
[2] IETF (http://www.ietf.org/), RFC 768.
[3] http://linux.about.com/od/commands/l/blcmdl8_netstat.htm

2. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.1. Przygotowanie laboratorium

W celu wykonania poniższych zadań laboratoryjnych należy:

1. Zalogować się w systemie Linux;
2. Pobrać odpowiednie pliki źródłowe z katalogu
   `Home/shared/pus/pus_02_udp;`

2.2. Zadanie 1. Komunikacja klient – serwer

W celu zapoznania się z podstawowymi funkcjami oprogramowania protokołu UDP stworzymy dwa programy komunikujące się ze sobą na zasadzie klient – serwer. Program klienta korzystając z protokołu UDP ustanawia połączenie z serwerem, serwer zaś przesyła informację, która w postaci czytelnej dla użytkownika określa wartości bieżące czasu i daty.

1. Utwórz katalog na pliki źródłowe, np. UDP:
   `$ mkdir UDP`
2. Skopiuj pliki źródłowe: *cw1_klient.c* oraz *cw1_serwer.c* do nowo utworzonego katalogu z katalogu: home/shared`/pus/pus_02_udp`
3. Skompiluj pliki źródłowe:
   `$ cc cw1_serwer.c –o serwer oraz cc cw1_klient.c –o klient`
4. Uruchom dwie konsole
5. Na pierwszej konsoli uruchom program serwer a na drugiej klient:
   `$ ./serwer oraz ./klient`
6. Zaobserwuj przepływ danych za pomocą programu ethereal.

Informacje dotyczące programu ethereal:

Nazwa	Opis
`ethereal`	Polecenie uruchamiające program
`Open w menu File`	Wczytanie pliku z zapisanymi ramkami
`Start i Stop z menu Capture`	Rozpoczynanie i kończenie przechwytywania ramek
`Capture Filters z menu Edit`	Dodawanie filtrów
`Add expression`	Ułatwia wprowadzanie reguł przy dodawaniu filtrów

Przydatne reguły:

Nazwa	Opis
`ip.src == adresIP`	Wyświetla pakiety IP pochodzące od adresIP
`ip.dst == adresIP`	Wyświetla pakiety IP skierowane do adresIP
`tcp.port`	Wyświetla pakiety z protokołem określonym przez numer portu

2.3. Zadanie 2. Klient i serwer echa za pomocą protokołu UDP

Zadaniem jest analiza przykładowego programu klienta i serwera. W tym przypadku jest to klient i serwer echa.
Do tego ćwiczenia potrzebne będą źródła programu które należy ściągnąć z katalogu:

```
$ cd  home/shared/pus/pus_02_UDP
```

Następnie należy skopiować pliki cw2_*klient.c* oraz cw2_*serwer.c* do katalogu np. „udp” na swoim koncie.
W celu uruchomienia programów należy wykonać następujące czynności:

1. Przejdź do katalogu ze źródłami serwera:
   `$ cd udp/`
2. Skompiluj źródła do postaci binarnej:
   `$ gcc –o cw2_klient cw2_klient.c`


```
$ gcc –o cw2_serwer cw2_serwer.c
```

3. Uruchom serwer na porcie 8500:
   ```
   $ ./cw2_server <nr_portu>
   ```
4. Uruchom klienta i przeanalizuj działanie programu
   ```
   $ ./cw2_klient <ip> <nr_portu>
   ```

2.4. Zadanie 3. Modyfikacja poprzednich programów

Zadaniem jest modyfikacja programów wcześniejszego ćwiczenia w taki sposób by serwer dostając od klienta dowolny tekst odsyłał go z powrotem odwracając litery i zamieniając je na duże znaki.

2.5. Zadanie 4. Kojarzenie adresu zdalnego z gniazdem

Zmodyfikuj program z ćwiczenia numer 2, tak aby można było używać funkcji recv()/send() w miejsce recvfrom()/sendto(). Jest to możliwe przy użyciu funkcji

```
#include <sys/socket.h>

int connect (s, (struct sockaddr *)&saddr,sizeof(saddr))
```

Funkcja zwraca 0 w wypadku wykonania poprawnego lub -1 w wypadku błędu

Parametr/opcja	**Opis**
`int s`	deskryptor gniazda
`(struct sockaddr *)&saddr`	zawiera adres protokołowy nadawcy
`sizeof(saddr)`	Określa wielkośc struktury wskazywanej poprzednim argumentem

Działanie jej opiera się na tym, że system automatycznie wypełnia parametry *from i *fromlen podanymi dla wywołania funkcji connect
Po skojarzeniu adresu zdalnego gniazdem lokalnym odbieranie danych możliwe jest za pomocą funkcji :

```
#include <sys/socket.h>

ssie_t recv(int sockfd, void *buff, size_t nbytes, int flags)
```


Parametr	Opis
sockfd	deskryptor gniazda
*buff	wskaźnik do bufora, do którego pobiera się dane
nbytes	liczba pobranych bajtów
flags	sygnalizatory

Do wysyłania danych służy funkcja :

```
#include <sys/socket.h>

ssie_t send(int sockfd, const void *buff, size_t nbytes, int flags)
```

Parametr	Opis
sockfd	deskryptor gniazda
*buff	wskaźnik do bufora, z którego wysyła się dane
nbytes	liczba odesłanych bajtów
flags	sygnalizatory

2.6. Zadanie 5. Brak sterowania przepływem w protokole UDP

Celem tego ćwiczenia jest przeanalizowanie skutku, że oprogramowanie UDP nie steruje przepływem danych.
Program działa w taki sposób, że klient wysyła ustaloną liczbę datagramów. W naszym przypadku klient odsyła do serwera 8000 datagramów UDP zawierających po 1400 bajtów. Zadaniem serwera jest odbieranie datagramów i ich zliczanie. Po przerwaniu pracy serwera poleceniem CRTL+C serwer wydrukuje odpowiedź ile odebrał datagramów i zakończy swoją pracę.

1. Przejdź do katalogu ze źródłami serwera:
   ```
   $ cd udp/
   ```
2. Skompiluj źródła do postaci binarnej:
   ```
   $ . Makefile
   ```
3. Uruchom serwer:
   ```
   $ ./cw5_server
   ```


4. Uruchom klienta i przeanalizuj działanie programu

```
$ ./cw5_klient <IP>
```

5. Po zakończeniu pracy klienta na konsoli gdzie został uruchomiony program serwera wciśnij CRTL +C

Przykładowy wynik programu:

$ odebrano 3036 datagramów

Klient wysłał 8000 datagramów, ale serwer otrzymał tylko 3036 datagramów, czyli reszta pakietów zaginęła. Oprogramowanie UDP nie powiadomiło w żaden sposób ani programu użytkowego serwera, ani programu użytkowego klienta o tym, że te datagramy zaginęły.

Uruchom kilka razy programy serwer i klienta i na podstawie obserwacji opisz w sprawozdaniu własne wnioski wypływające z tego prostego doświadczenia. W celu dojścia do ciekawych wniosków zaobserwuj za pomocą programu netstat ile datagramów dotarło do maszyny serwera.

3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. *„Protokół UDP”* polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego dostarczonego na kolejne zajęcia licząc od daty laboratorium, kiedy zadania zostały zadane.
Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- trudniejsze kawałki kodu, opisane tak, jak w niniejszej instrukcji,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/realizowalne podczas zajęć, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.

