ZESZYTY ETI ZESPOŁU SZKÓŁ W TARNOBRZEGU
Nr 1 Seria: Teleinformatyka 2013

Paweł Kowalik

Zespół Szkół im. ks. S. Staszica w Tarnobrzegu

KOMUNIKACJA SIECIOWA MIĘDZY URZĄDZENIAMI Z WYKORZYSTANIEM PROTKOŁU TCP W SYSTEMACH UNIX

Streszczenie

W pracy przedstawiony został cały proces sieciowej wymiany danych pomiędzy połączonymi ze sobą urządzeniami zarówno w sieci lokalnej, jak i w Internecie. Komunikacja odbywa się przy użyciu protokołu TCP a każde z urządzeń działa w systemie Unix, zatem połączenie odbywać się będzie przy użyciu gniazd, które także zostały wyjaśnione. Ponadto praca zawiera przykładowe gotowe do użycia programy napisane w języku C realizujące proces połączenia i wymiany danych.

1. WSTĘP

Działanie sieci lokalnej oraz Internetu opiera się na komunikacji pomiędzy połączonymi ze sobą urządzeniami w celu wymiany potrzebnych informacji. Same urządzenia jednak nie są w stanie nawiązać ze sobą połączenia sieciowego bez odpowiedniego oprogramowania. Na przykładzie komputera takim oprogramowaniem może być komunikator, przeglądarka WWW czy klient poczty e-mail (oczywiście wraz z systemem operacyjnym). Każda z tych aplikacji umożliwia komunikację poprzez sieć z inną aplikacją działającą na innym urządzeniu. W przypadku komunikatora będzie to realizacja połączenia w celu przesyłania wiadomości między użytkownikami, przeglądarka WWW umożliwi nam odebranie informacji z interesującej nas strony internetowej, zaś za pomocą klienta poczty e-mail będziemy mogli wysyłać i odbierać pocztę elektroniczną. Wszystkie te czynności mogą odbywać się dzięki nawiązywaniu odpowiednich połączeń sieciowych, które nierzadko docierają na drugi koniec kuli ziemskiej w przeciągu kilkudziesięciu milisekund. Przyjrzyjmy się zatem w jaki sposób realizowane jest takie połączenie na przykładzie systemów UNIX.

2. PROCEDURA NAWIĄZANIA POŁĄCZENIA SIECIOWEGO

Zanim przejdziemy do zapoznania się ze szczegółowym procesem nawiązania połączenia sieciowego, należy wyjaśnić kilka kwestii. Jak już wspomniałem wcześniej, będziemy korzystać z protokołu TCP (**Transmission Control Protocol)** w systemie Unix. Protokół ten gwarantuje nam niezawodność połączenia i zapewnia, że wysłane przez nas dane dotrą do celu dokładnie w takiej kolejności, w jakiej „opuściły" nasze urządzenie (w innym wypadku zgłoszony zostanie błąd). Teraz zajmijmy się kolejnym zagadnieniem jakim są gniazda w systemach Unix.

2.2 Pojęcie gniazda (socket)

Gniazda w systemie Unix są podstawą do nawiązania komunikacji z innym urządzeniem. Są one punktami końcowymi połączenia pozwalającego na przesyłanie danych w obie strony pomiędzy dwoma aplikacjami działającymi w sieci. Aby zrozumieć ich działanie w praktyce, po pierwsze należy wiedzieć że w systemie Unix wszystko jest plikiem, tzn. że każda operacja wejścia/wyjścia odbywa się poprzez odczytywanie lub zapisywanie do pliku. Nie inaczej jest z gniazdami, więc jeśli zajdzie konieczność wysłania lub odebrania danych poprzez sieć, będzie konieczne odpowiednio zapisanie lub odczytanie danych z pliku. Cały proces odbywać się będzie przy użyciu **deskryptorów plików.** Są to unikalne liczby całkowite, z których każda wskazuje na konkretny otwarty plik w systemie. W programowaniu deskryptor jest tworzony automatycznie podczas otwierania pliku i jest on wykorzystywany do wszystkich operacji z nim związanych. Jeżeli więc

będziemy chcieli wykonać jakiekolwiek zadanie na pliku, używamy jego deskryptora stworzonego automatycznie podczas otwarcia pliku.

W zależności od używanego protokołu istnieje wiele typów gniazd, z których możemy skorzystać. Dla protokołu TCP będzie to gniazdo strumieniowe (stream socket) i z tego typu gniazda będziemy właśnie korzystać.

Wiemy już zatem, że do nawiązania naszego połączenia będzie potrzebne utworzenie specjalnych gniazd (czyli po prostu plików), jednak oprócz tego konieczne będą jeszcze 2 dodatkowe informacje. Mianowicie jest to adres IP oraz numer portu TCP.

2.3 Proces połączenia

Aby jedno urządzenie mogło nawiązać połączenie z drugim, musi znać jego **adres IP oraz numer portu**, na którym działa aplikacja, z która chcemy się połączyć. Na tym etapie konieczne jest już określenie funkcji poszczególnych urządzeń. Pierwsza z nich to **klient,** który będzie chciał połączyć się z drugą maszyną, która to z kolei będzie pełnić funkcję **serwera,** czyli ciągłego oczekiwania na nadchodzące połączenie oraz odebranie (lub odrzucenie) go, jeśli takie nastąpi. Klient w celu otwarcia połączenia potrzebuje adresu IP serwera, czyli konkretnego identyfikatora sieciowego, pod którym on występuje w sieci. Oprócz tego potrzebny będzie także numer portu TCP, pod którą działa aplikacja, z którą chcemy się połączyć. Jest on konieczny z tego względu, że serwer może posiadać wiele aplikacji udostępnionych w sieci, np. serwer WWW, poczty, FTP, więc sam adres IP nie wystarcza aby określić, do której z nich chcemy się dostać.

Port Name	Port Number /Protocol	Alias
ftp	21/tcp	--
telnet	23/tcp	--
smtp	25/tcp	mail
nicname	43/tcp	whois
domain	53/tcp	nameserver
domain	53/udp	nameserver
finger	79/tcp	--
http	80/tcp	www www-http
pop3	110/tcp	pop-3
auth	113/tcp	authentication tap ident
nntp	119/tcp	readnews untp
ntp	123/udp	--
https	443/tcp	--

Tabela 1 – Najczęściej używane porty standardowych usług (serwer HTTP, FTP, SMTP)

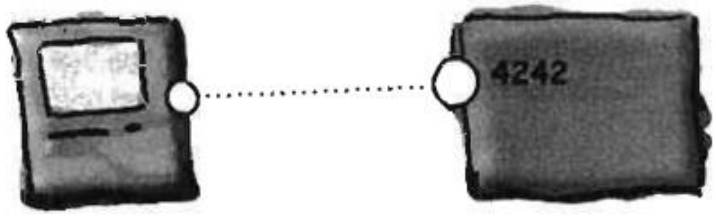
(rys. informit.com)

Mamy już zatem wszystkie niezbędne informacje potrzebne klientowi do połączenia się z serwerem, czyli adres IP, numer portu oraz gniazdo, przez które połączenie zostanie nawiązane. Przejdźmy zatem do zapoznania się z samym procesem połączenia w praktyce.

1. Pierwszy działanie rozpoczyna serwer, który rozpoczyna **nasłuchiwanie** na połączenia przychodzące od klientów na konkretnym porcie (tworzy gniazdo serwera). Dopiero teraz klient jest w stanie nawiązać połączenie, a dopóki do niego nie dojdzie, w tym momencie rola serwera się kończy.

Rysunek 1 – Rozpoczęcie nasłuchiwania na porcie 4242 przez serwer (po lewej – klient)

2. W kolejnym kroku klient próbuje połączyć się z serwerem używając jego adresu IP oraz portu, na którym serwer nasłuchuje. W tym celu tworzy po swojej stronie gniazdo o takim samym typie jak serwer, czyli strumieniowe, ponieważ korzystamy z protokołu TCP. Przez to gniazdo klient będzie komunikował się z serwerem.



Rysunek 2 – Klient tworzy gniazdo i próbuje nawiązać połączenie z serwerem

3. W momencie gdy serwer napotka na przychodzące połączenie, **może** je zaakceptować. Jeśli tak się stanie, w tym momencie oba urządzenia już dysponują informacjami o sobie (adres IP, numer portu) i mogą przesyłać między sobą informacje. Warto podkreślić, że połączenie nie zostało zrealizowane na porcie, na którym działa serwer, lecz na pierwszym innym dostępnym porcie serwera po to, aby mógł on nasłuchiwać na kolejne połączenia od kolejnych klientów na jednym i tym samym porcie. **Podłączenie każdego kolejnego klienta wymaga użycia kolejnego portu i tym samym utworzenia kolejnego gniazda przez serwer.**

Rysunek 3 – Serwer akceptuje połączenie i tworzy dla niego kolejne gniazdo na kolejnym porcie

2.4 Podsumowanie

W ten sposób odbywa się cały proces nawiązania połączenia sieciowego w teorii. Wiadomo już, że aby do takiego połączenia doszło, oba urządzenia muszą o sobie „wiedzieć" tj. znać adres IP identyfikujący urządzenie w sieci oraz numer portu, czyli konkretnej usługi, z której klient chce skorzystać. Cały proces

odbywa się za pomocą utworzonych wcześniej gniazd zarówno po stronie klienta jak i serwera. Klient próbuje nawiązać połączenie wykorzystując adres IP oraz port serwera, którego to z kolei zadaniem jest ciągłe nasłuchiwanie na połączenia, czyli zaakceptowanie (odrzucenie) go, jeśli takie nastąpi.

3. REALIZACJA POŁĄCZENIA SIECIOWEGO W PRAKTYCE

W tym rozdziale zostały zaprezentowane przykładowe programy wraz z wyjaśnieniami napisane w języku C realizujące proces prostego połączenia pomiędzy dwoma urządzeniami na zasadzie klient-serwer. Potrzebne będą zatem dwie aplikacje – jedna służąca jako serwer do nasłuchiwania na połączenia, oraz druga pełniąca rolę klienta w celu nawiązania połączenia z serwerem.

3.1 Przykładowa aplikacja serwerowa

Na początku przedstawiony zostanie przykładowy program realizujący pracę prostego serwera. Będzie miał on za zadanie nasłuchiwać na połączenia przychodzące na wybranym porcie, a w momencie gdy takie połączenie nastąpi, zaakceptować je. Następnie klient będzie miał możliwość wysyłania wiadomości do serwera, który będzie zarówno wyświetlał je u siebie jak i odsyłał z powrotem do klienta (echo).

Program należy rozpocząć od dołączenia potrzebnych plików nagłówkowych do programu:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
```

Plik **stdio.h** pozwala korzystać z funkcji wejścia/wyjścia, a konkretniej z funkcji **printf()**, która będzie wypisywać wiadomość przesłaną do klienta przez serwer oraz informacje pomocnicze odnośnie procesu połączenia. Ostatnie 3 nagłówki umożliwiają użycie wszelkich typów zmiennych oraz funkcji niezbędnych do pracy z gniazdami oraz do realizacji połączenia zarówno po stronie klienta jak i serwera. Taki sam zestaw czterech plików nagłówkowych będziemy wykorzystywać w aplikacji klienta.

Następnie zostanie zadeklarowana główna funkcja programu – **main()**, w której będzie znajdować się cała funkcjonalność serwera. Na początku zostały również zdefiniowane wszystkie zmienne, które zostaną użyte w programie:

```
...
int main(int argc, char *argv[]) {
    int gniazdoSerwera, portSerwera;
    struct sockaddr_in serwer;
    char bufor[64];
...
```

Parametr **argv** funkcji **main()** przechowuje informacje przekazane programowi jako tekst podczas jego uruchomienia. W tym przypadku będzie używana tylko jedna informacja – port na którym będzie działać serwer. Na samym początku funkcji następuje deklaracja zmiennych. Pierwsza z nich – **gniazdoSerwera** – jak sama nazwa wskazuje będzie przechowywać gniazdo, poprzez które serwer będzie odbierał połączenia od klientów. Zmienna **portSerwera** będzie zawierała numer portu TCP (już nie w formie tekstu, a przekonwertowany na liczbę całkowitą), na którym zostanie uruchomiony serwer. Kolejna zmienna – **serwer** – jest strukturą (zbiorem wielu zmiennych różnego typu) zawierającą wszystkie informacje na temat samego serwera, tj. przypisany adres IP, jego typ (IPv4 lub IPv6) oraz port. Ostatnią zmienną (tablicą) jest **bufor**. Pomieści ona 64 bajty i będzie służyć do przechowywania odebranych wiadomości od klienta.

3.1.1 Tworzenie gniazda

Po zdefiniowaniu odpowiednich zmiennych możliwe jest już utworzenie gniazda serwera i przypisanie go do wcześniej zadeklarowanej zmiennej:

```
...
gniazdoSerwera = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
...
```

Jak widać gniazdo tworzone jest za pomocą funkcji **socket(),** do której należy przekazać 3 parametry. Pierwszy z nich to typ wykorzystywanych adresów IP. Połączenie będzie realizowane z wykorzystaniem adresów IPv4, czyli standardowych 4 bajtów oddzielonych kropkami, np. 208.67.220.220. Temu typowi adresów odpowiada wartość **AF_INET**. Drugi parametr to typ gniazda (patrz rozdz. 2.2). Korzystając z gniazd strumieniowych należy użyć wartości **SOCK_STREAM.** Ostatnim parametrem jest używany protokół, czyli w tym przypadku TCP, któremu odpowiada wartość **IPPROTO_TCP.**

3.1.2 Przypisanie adresu IP i rozpoczęcie nasłuchiwania

Samo utworzenie gniazda nie wystarcza, aby serwer mógł rozpocząć pracę. Nadal nie „wie" on jeszcze na jakim adresie IP i porcie ma nasłuchiwać. W tym celu należy wypełnić wcześniej zadeklarowaną strukturę **serwer** oraz użyć funkcji **bind()** w celu przypisania serwerowi potrzebnych informacji:

```
...
serwer.sin_family = AF_INET;
serwer.sin_addr.s_addr = htonl(INADDR_ANY);
serwer.sin_port = htons(portSerwera);
bind(gniazdoSerwera, (struct sockaddr *)&serwer, sizeof(serwer));
...
```

Struktura **serwer** została wypełniona następującymi danymi. Pole **sin_family** określa typ adresów IP podobnie jak w przypadku tworzeniu gniazda. Następnie **s_addr** określa adres, na którym będzie nasłuchiwał serwer. Tutaj została podana wartość **INADDR_ANY**, która jest równoznaczna z adresem IP 0.0.0.0. Oznacza to, że serwer będzie odbierał połączenia na wszystkich możliwych adresach IP, jakie zostały u niego skonfigurowane. Ostatnie pole - **sin_port** - jak sama nazwa wskazuje określa port nasłuchiwania serwera. Następnie została użyta funkcja bind w celu przypisania serwerowi wszystkich trzech informacji.

Jeśli wszystkie powyższe polecenia wykonają się poprawnie, w tej chwili serwer może już rozpocząć nasłuchiwanie. Do tego posłuży funkcja **listen():**

```
...
listen(gniazdoSerwera, 5);
...
```

Jako pierwszy parametr podane zostało utworzone wcześniej gniazdo, zaś drugi parametr określa maksymalną liczbę oczekujących połączeń od klientów, w tym przypadku 5.

3.1.3 Akceptowanie połączeń, wysyłanie i odbieranie danych

Po wykonaniu funkcji listen() serwer rozpoczyna nasłuchiwanie, czyli oczekiwanie na przychodzące połączenia od klientów. Aby jednak do takiego połączenia mogło dojść, serwer musi je najpierw zaakceptować. W tym celu wywołana zostaje funkcja **accept(),** jednak wcześniej potrzebnych będzie kilka nowych zmiennych:

```
...
struct sockaddr_in klient;
int gniazdoKlienta;
socklen_t klientRozmiar = sizeof(klient);
gniazdoKlienta = accept(gniazdoSerwera, (struct sockaddr *)&klient, &klientRozmiar);
...
```

Struktura **klient** będzie przechowywać informacje o połączonym kliencie, m.in. adres IP oraz port na którym ustanowiono połączenie (inny niż port nasłuchiwania serwera). Potrzebna będzie także zmienna **gniazdoKlienta,** przechowująca gniazdo poprzez które będą wysyłane i odbierane dane. W tej zmiennej zapisywany jest wynik funkcji accept(), co oznacza że zwraca ona utworzone gniazdo do komunikacji z klientem.

Po zaakceptowaniu połączenia możliwa jest już komunikacja między klientem a serwerem. Do tego celu służą dwie podstawowe funkcje: **send()** do wysyłania oraz **recv()** do odbierania danych:

```
...
send(gniazdoKlienta, bufor, 64, 0);
recv(gniazdoKlienta, bufor, 64, 0);
...
```

Jak widać obie z nich wymagają takich samych parametrów. Pierwszy z nich to gniazdo, przez które będą odpowiednio wysyłane lub odbierane informacje. Kolejny parametr wskazuje na dane, które mają zostać wysłane lub miejsce gdzie mają być odebrane. Do tego celu używamy utworzonej na samym początku zmiennej bufor o wielkości 64 bajtów, którą to z kolei wielkość należy podać jako następny parametr. Jako ostatni parametr podawane są tzw. flagi czyli opcje dotyczące sposobu odbierania i wysyłania danych. Nie są one w tym momencie istotne, zatem wpisane zostało po prostu 0.

Dzięki tym funkcjom zarówno serwer jak i klient mogą już odbierać i wysyłać do siebie wiadomości.

Aby zakończyć połączenie z klientem, serwer może wywołać funkcję **close()**, która jako swój jedyny parametr przyjmuje gniazdo na którym ustanowiono połączenie, czyli w tym przypadku może być to zmienna gniazdoKlienta. W taki sam sposób serwer może również zakończyć swoją pracę i przestać przyjmować połączenia. W tym celu jako parametr należy podać gniazdo, na którym nasłuchuje serwer, czyli zmienną gniazdoSerwera.

3.1.4 Podsumowanie

Przykład najprostszej aplikacji serwerowej z wykorzystaniem protokołu TCP opiera się na wykonaniu kilku niezbędnych kroków. Pierwszy z nich to utworzenie odpowiedniego typu gniazda za pomocą funkcji socket(), na którym będą przyjmowane połączenia przychodzące. Następnie korzystając z funkcji bind() serwerowi zostają przypisane informacje z wcześniej wypełnionej struktury o tym, na jakim adresie IP oraz porcie ma rozpocząć nasłuchiwanie, czyli kolejny etap swojej pracy. Rozpoczęcie nasłuchiwania odbywa się przy użyciu funkcji listen(). Kolejnym krokiem jest zaakceptowanie nadchodzącego połączenia od klienta. Do tego celu służy funkcja accept(), która wymaga utworzenia nowego gniazda na nowym porcie dla klienta, przez które będą przesyłane dane za pomocą funkcji send() oraz recv(). Serwer może zakończyć połączenie z klientem jak i również zakończyć proces nasłuchiwania przy użyciu funkcji close().

3.2 Przykładowa aplikacja klienta

W celu połączenia się z wcześniej przygotowaną aplikacją serwera, zostanie przedstawiona prosta aplikacja klient, która oprócz samego połączenia się z serwerem będzie miała za zadanie wysyłać do niego wiadomości wpisywane przez użytkownika aplikacji.

Na początku należy dołączyć poznane wcześniej (rozdz. 3.1) takie same jak w przypadku serwera pliki nagłówkowe. Następnie w deklaracji funkcji main() będą znajdować się zmienne również podobne do tych użytych w aplikacji serwera, jednak z inną nazwą jako że jest to aplikacja pełniąca rolę klienta:

```
...
int gniazdoKlienta, portSerwera;
char bufor[64];
struct sockaddr_in serwer;
...
```

Za pomocą zmiennej **gniazdoKlienta** nastąpi zarówno połączenie z serwerem jak i wymiana z nim danych, w tym przypadku wiadomości tekstowych. Jako parametry programu zostaną przekazane adres IP serwera, z którym należy się połączyć oraz port, na którym serwer nasłuchuje. Tablica **bufor** będzie przechowywać wiadomości pobrane od użytkownika za pomocą klawiatury w celu ich późniejszego wysłania do

serwera. Struktura **serwer** natomiast będzie zawierać informacje o serwerze niezbędne do nawiązania z nim połączenia. Należy je zatem najpierw poprawnie uzupełnić, a wcześniej utworzyć także gniazdo dla komunikacji z serwerem:

```
...
gniazdoKlienta = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
serwer.sin_family = AF_INET;
inet_addr(argv[1], &serwer.sin_addr.s_addr);
serwer.sin_port = htons(portSerwera);
...
```

Gniazdo klienta musi być takiego samego typu jak w przypadku serwera, zatem wykorzystane zostały identyczne parametry, czyli typ adresów IP: IPv4, typ gniazda: strumieniowe, protokół: TCP. Następnie został określony typ adresów IP odpowiadający serwerowi, czyli AF_INET (IPv4). W kolejnym kroku wykorzystana została funkcja **inet_addr()** w celu pobrania adresu IP serwera przekazanego jako parametr przy uruchomieniu programu oraz zapisania go w poprawnej formie w polu s_addr. Na końcu uzupełniony został port serwera również przekazany jako parametr aplikacji.

3.2.1 Nawiązanie połączenia z serwerem

Teraz możliwe jest już ustanowienie połączenia z działającym serwerem. Do tego celu służy funkcja **connect():**

```
...
connect(gniazdoKlienta, (struct sockaddr *)&serwer, sizeof(serwer));
...
```

Funkcja ta przyjmuje 3 parametry: gniazdo przez które zostanie nawiązane połączenie, wypełnioną wcześniej strukturę serwer m.in. z informacjami o adresie IP i porcie serwera oraz rozmiar tej struktury uzyskany za pomocą funkcji **sizeof()**.

Jeśli wszystkie polecenia wykonają się poprawnie i nie zostaną zwrócone błędy, w tym momencie zostało już nawiązane połączenie z serwerem i istnieje możliwość wymiany danych. Do tego celu posłużą oczywiście wcześniej przedstawione (rozdz. 3.1.3) funkcje send() oraz recv().

Wiadomości będą wpisywane przez użytkownika na klawiaturze i pobierane za pomocą funkcji **fgets():**

```
...
fgets(bufor, 64, stdin);
...
```

Pierwszy parametr określa miejsce do którego zostanie wczytana wiadomość, do czego posłuży utworzony na samym początku bufor. Drugi parametr to wielkość tego bufora, natomiast **stdin** podany jako trzeci i ostatni argument oznacza najprościej mówiąc, że tekst będzie wczytywany z klawiatury użytkownika (standardowy strumień wejścia).

3.2.2 Podsumowanie

Proces połączenia się klienta z serwerm odbywa się w znacznie prostszy sposób niż w przypadku obsługi pracy serwera. Do tego celu wystarczy bowiem utworzenie gniazda, przekazanie informacji o serwerze, z którym będzie nawiązywane połączenie do odpowiedniej struktury oraz przekazanie tej struktury do funkcji connect(), która ustanawia gotowe połączenie. Wysyłanie i odbieranie danych odbywa się za pomocą takich samych funkcji jak w przypadku serwera, czyli send() oraz recv().

4. ZAKOŃCZENIE

Komunikacja w sieci lokalnej oraz w sieci Internet wymaga (najczęściej) nawiązywania połączeń, które w przypadku Internetu ustanawiane są praktycznie nieustannie przez miliardy urządzeń do niego podłączonych. Do utworzenia takiego połączenia najczęściej obecnie stosowanym protokołem jest TCP, który umożliwia bezproblemowe przesyłanie danych w sposób strumieniowy. W systemach Unix przeważnie wszystkie pliki nagłówkowe niezbędne do napisania prostych aplikacji pełniących rolę klienta lub serwera, czyli odpowiednio

urządzenia które chce nawiązać połączenie oraz urządzenia które na połączenia oczekuje (nasłuchuje). Pliki te umożliwiają skorzystanie z funkcji takich jak socket(), bind(), listen(), accept(), connect() operujących na gniazdach, czyli końcowych punktach połączenia przez które następuje komunikacja w obie strony. Ponadto do dyspozycji są również funkcje pozwalające na wysyłanie i odbieranie danych poprzez aktywne połączenie – send() oraz recv().

BIBLIOGRAFIA

[1] **Beej's Guide to Network Programming** - http://beej.us/guide/bgnet/, 2012
[2] "**Head First Java" - Kathy Sierra, Bert Bates,** 2005

NETWORK COMMUNICATION BETWEEN DEVICES USING TCP PROTOCOL IN UNIX SYSTEMS

Summary

This document describes the entire network data exchange process between devices connected to each other with LAN network as well as with the Internet. The communication is based on TCP protocol and the devices are working under Unix-like system. The document also contains descriptions of applications written in C language implementing the network communication process.