Eduards Šarņeckis
Automātika un datortehnika 2. grupa

3. atskaite

LPC-2478-STK izstrādes plates savienojums ar datoru, izmantojot UPD un TCP protokolus

Teorētiskā daļa

Ligzda (Socket)

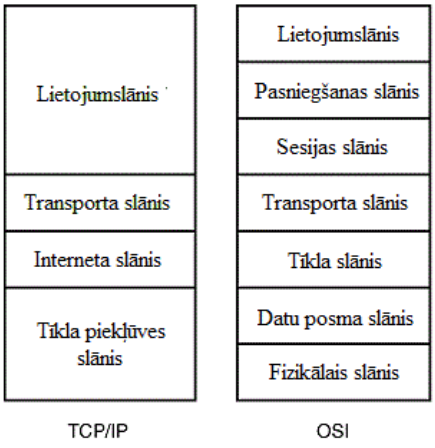
Tīkla ligzda ir starpprocesu komunikācijas plūsmas galapunkts datoru tīkla. Mūsdienās komunikācija starp datoriem tiek nodrošināta, izmantojot interneta protokolu IP. Ligzdas lietojumprogrammas saskarni parasti nodrošina operētājsistēma, un tas atļauj datorprogrammām izmantot tīkla ligzdas, kuras balstās uz Berkeley ligzdu standarta. Ligzdas adrese ir IP-adreses un porta kombinācija. Pamatojoties uz šo adresi, interneta ligzda piegādā ienākošas datu paketes uz atbilstošo lietojuma procesu vai pavedienu.

Eksistē 3 tipu tīkla ligzdas:

- Datagrammu ligzdas, kuras izmanto UDP protokols
- Plūsmas ligzdas, kuras izmanto TCP un SCTP protokoli.
- IP-ligzdas, kuras ir pieejamas maršrutētajos un citās tīkla iekārtās. Izmantojot šīs ligzdas, transporta slānis netiek izmantots un pakešu galvenes dati ir pieejami programmai.

UPD un TCP protokoli

UPD (lietotāja datagrammu protokols) un TCP (pārraides vadības protokols) ir OSI (atvērto sistēmu sadarbības bāzes etalonmodelis) transporta līmeņa protokoli, kas nodrošina datu pārsūtīšanu starp datoriem, izmantojot to IP-adreses. Parastais OSI modelis definē 7 slāņus, taču TCP/IP modelī var būt definēti tikai 5 vai 4 slāņi, jo pasniegšanas un sesijas slāņi netiek definēti vai ir apvienoti ar lietojumslāni (1.att.).



1.att. OSI un TCP/IP modeļu slāņi

Neskatoties uz to, ka gan UPD, gan TCP protokols pieder pie transporta slāņa un abi protokoli nodrošina datu pārsūtīšanu ziņojumu veidā, tiem ir ievērojamas atšķirības:

1) TCP ir savienojumu-orientēts protokols – tas nozīmē, ka starp abiem datoriem tiek uzstādīts savienojums. Lai uzstādītu savienojumu ar mērķi, tiek veikts tā saucamais trīspusīgs rokasspiediens. UPD sūta datagrammas bez savienojuma ar mērķi.
2) UPD ir ātrāks par TCP, taču tas nenodrošina datu nogādi līdz mērķim.
3) TCP protokols veic pārsūtītu datu pakešu sakārtošanu, kas nodrošina pareizu pakešu apstrādes secību, savukārt UPD gadījumā pakešu kārtošana (ja tā ir nepieciešama) ir jānodrošina lietojumslānī.
4) TCP dati tiek sūtīti neierobežota garuma straumes veida norīkotā savienojuma ietvaros, bet UPD gadījuma dati tiek sūtīti atsevišķu neliela izmēra pakešu veidu.
5) Abi protokoli nodrošina kļūdu kontroli, taču UPD protokoli nepiedāvā zaudēto datu atgūšanu un nenodrošina datu plūsmas kontroli, kas var novest pie zaudējumiem, ja tīkls ir pārslogots.
6) Izmantojot TCP protokolu, uz katru ziņojumu vai datu paketi mērķis atbildēs ar apstiprinājuma ziņojumu (ACK).

Sakarā ar šim īpašībām TCP protokoli tiek izmantoti lietojumos, kur nav nepieciešama ātra datu nogādāšana līdz mērķim, bet ir svarīga datu pakešu nonākšana līdz mērķim un pakešu secība. UPD tiek pielietots tīkla spēlēs un lietojumos, kuros ir nepieciešama ātra datu pārsūtīšana un zaudējumi nav kritiski darbībai. UDP var būt noderīgs serveros, kas apstrādā nelielus vaicājumus no vairākiem klientiem.

<u>TCP protokola piemēri:</u>

A) Internets (TCP Port 80)
B) E-pasts (SMTP TCP Port 25)
C) Datņu pārsūtīšanas protokols (FTP Port 21)
D) Secure Shell (OpenSSH Port 22)
E) Telnet

<u>UPD protokola piemēri:</u>

A) Domēnu nosaukumu sistēma (DNS UDP Port 53)
B) Video un filmu straumēšana
C) Voice over IP (VoIP)
D) Triviālu datņu pārsūtīšanas protokols (TFTP)
E) Interneta spēles

Praktiskā daļa

TFTP serveris, kas tika nokonfigurēts pirmā praktiskā darba gaitā, lai ielādētu uz LPC-2478-STK izstrādes plates uClinux failus un vienkāršu helloworld programmu, ir viens no UPD protokola izmantošanas piemēriem.

Šī praktiskā darba gaitā tika realizēts savienojums starp LPC-2478-STK izstrādes plates un host datoru, izmantojot Ethernet savienojumu, un TCP tiek izmantots kā savienojuma protokols. Savienojumam tika izmantota „Linux networking socket" slāņu pieeju, un pats savienojums tiek realizēts klients-serveris veidā. Šī pieeja darbojas tikai Linux operētājsistēmas vidē, tāpēc, ka datora operētājsistēma tiek izmantota Ubuntu. Winsocket pieeja, kas var tikt izmantota Windows vidē, ir programmējama citādi.

Lai realizētu savienojumu starp datoru un izstrādes plati, tika izmantotas divas C programmēšanas valodā uzrakstītas programmas – viena reprezentē klienta daļu, un otra ir servera daļa. Gan plati, gan datoru ir iespējams izmantot kā klientu vai serveri. Lai nokompilētu serveri un klientu priekš palaišanas uz LPC-2478-STK bija nepieciešams izmantot kroskompilatoru arm-linux-gcc. Iepriekšējā darba gaita jau tika nokompilēta helloworld programma, tāpēc tika izmantots jau nokonfigurētais Makefile no iepriekšējā darba. Abu programmu pirmkodus ar komentāriem ir iespējams apskatīt darba pielikumā.

Lai nokompilētu klientu un serveri palaišanai uz datora Ubuntu operētājsistēmas vidē nav jāizmanto var izmantot parastu GCC kompilatoru. Lai nokompilētu programmas, ir jāatver terminmāls un ar **cd** komandas palīdzību jāparej uz direktoriju, kur atrodas client.c un server.c faili. Tad var izpildīt kompilēšanu:

gcc -o client client.c

gcc -o server server.c

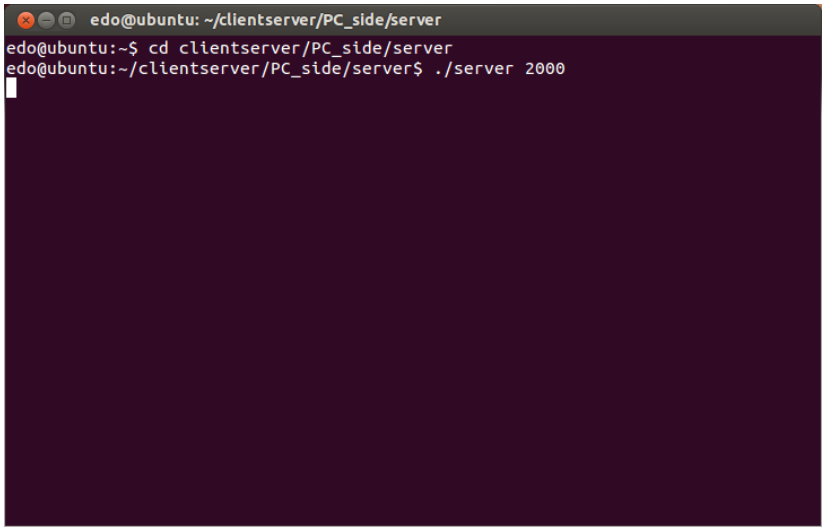
Kompilēšana palaišanai uz LPC-2478-STK notiek izmantojot Makefile. client.c un server.c priekš plates atrodas dažādās direktorijas. Lai tos nokompilētu, ir jāpalaiž **make** komanda, katra no abām direktorijām. Kad abas daļas ir veiksmīgi nokompilētas, tas var ielādēt plate un veidot sakarus.

Izstrādes plati var izmantot gan ka klientu, no kura tiks padots ziņojums, gan ka serveri, kas saņems ziņojumu un atsutis klientam apstiprinājuma ziņojumu. Praktiski tika izmēģināti abi varianti, taču šeit ir aprakstīts tikai viens variants – LPC-2478-STK plate ir klienta puse, un dators ir serveris – jo otrs variants, kad plate kalpo par serveri, ir ļoti līdzīgs. Viss, kas atšķiras, ir ielādējama uz izstrādes plates programma un palaišanas kārtība (sākuma vienmēr ir jāpalaiž serveris).

Servera palaišanai ir jāizmanto kāds no portiem diapazona no 2000 līdz 65535:

./server 2000

Serveris ir palaists un klausās 2000 portu (2.att).



2.att. Servera palaišana uz datora

Darba ietvaros uClinux tiek ielādēts uz LPC-2478-STK no USB flash kartes un palaists automātiski, tāpēc bija nepieciešams tikai nokonfigurēt Ethernet:

ifconfig eth0 192.168.1.20

Kad Ethernet ir nokonfigurēt ir nepieciešams pāriet uz var direktoriju: **cd var**

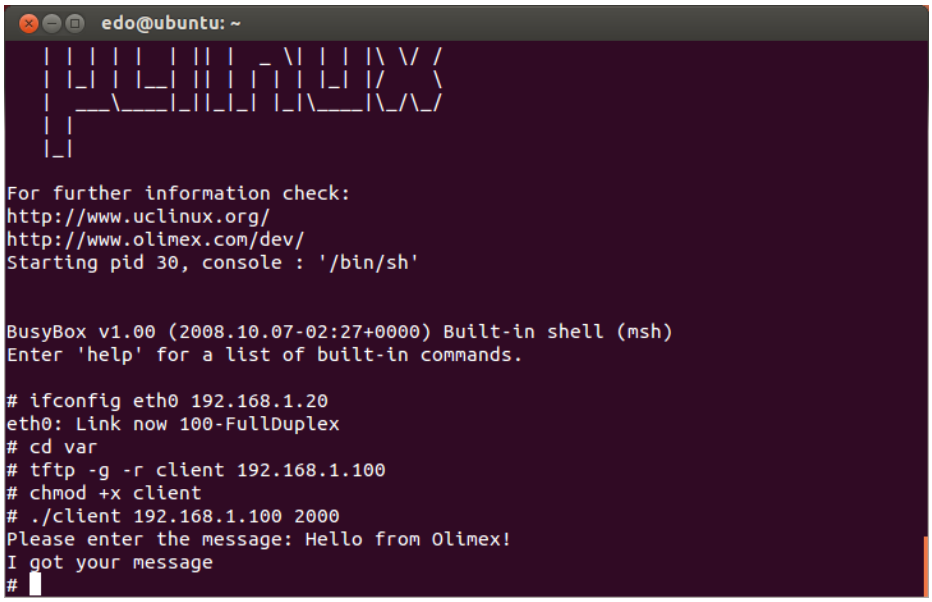
Tagad var ielādēt client programmu no datora TFTP servera un palaist to:

tftp –g –r client 192.168.1.100

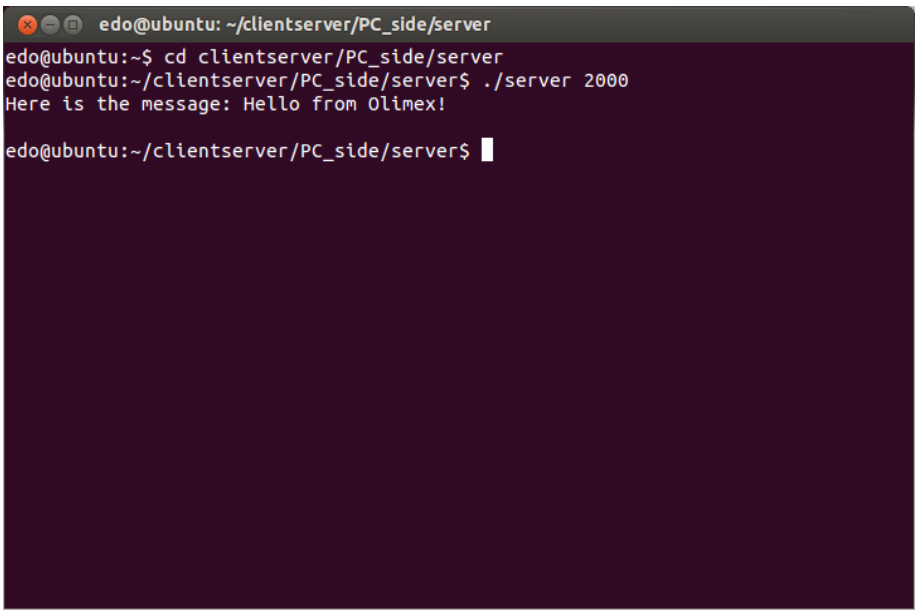
chmod +x client

./client 192.168.1.100 2000

Lai palaistu klientu ir, jānorāda servera hostname (nokonfigurētu datora nosaukumu) vai IP-adresi un portu. Jā savienojums ar serveri ir veiksmīgi, tad klienta pusei tiek piedāvāts ievadīt paziņojumu, kas tiks nosūtīts serverim. Ja serveris saņems ziņojumu, tad tas tiek izvadīts terminālā un klients saņem apstiprinājuma paziņojumu (3.att. un 4.att.).



3.att. Klients sūta paziņojumu un saņem atbildi no servera



4.att. Serveris saņem paziņojumu no klienta sūta apstiprinājumu

Pēc komunikācijas savienojums tiek aizvērts un klients un serveris beidz savu darbu.

Programmu koda apraksts

Lai varētu programmēt ligzdas, izmantojot C programmēšanas valodu, ir nepieciešamas pieslēgt attiecīgos iesākumfailus:

- **#include <sys/socket.h>.** Pieslēdz socket.h iesākumfailu, kas satur vairākas definētas vērtības un struktūras, lai varētu strādāt ar ligzdām.
- **#include <netinet/in.h>.** Pieslēdz in.h iesākumfailu, kas satur konstantes un struktūras, kas nepieciešamas, lai apstrādātu interneta domēnu adreses.
- **#include <sys/types.h>.** Pieslēdz types.h iesākumfailu, kas satur datu struktūru definīcijas, kas tiek izmantotas sistēmas izsaukumiem. Šie datu veidi tiek pielietoti iepriekšējos iesakumfailos.

Gan servera daļa, gan klienta daļa tiek izmantota funkcija **void error(char *msg)**. Šī funkcija apstrādā sistēmas izsaukumu kļūdas, izvada paziņojumu par kļūdu un pabeidz programmas darbību.

Servera daļas **main** funkcijas argumenti **int argc** un **char *argv[]**, prasti netiek izmantoti. Tie ir īpaša veida argumenti, kas dod iespēju nodot programmai vērtības, ja programma tiek palaista no komandas rindas. Piemēri:

```
server 2000
client 127.0.0.1 2000
```

Skaitlis 2000 norāda portu un 127.0.0.1 norāda servera IP-adresi.

`struct sockaddr_in, serv_addr, cli_addr` ir struktūras, kas ir definētas in.h iesākumfailā: ligzdas adreses struktūra, servera adreses struktūra un klienta adreses struktūra.

```
if (argc < 2)
{
 fprintf(stderr,"ERROR, no port provided");
 exit(1);
}
```

Šī IF funkcija apstrādā datus, kurus lietotājs ievada, palaižot programmu no komandas rindas, un izdod kļūdas paziņojumu, ja servera komunikācijas ports netika norādīts.

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

Tiek izveidota jauna komunikācijas ligzda. Funkcijai `socket` ir 3 argumenti:

1) Ligzdas adreses domēns, kas dotajā gadījumā ir `AF_INET` jeb internets
2) Ligzdas tips var būt `SOCK_STREAM` (datu plūsmas) vai `SOCK_DGRAM` (datagrammu).
3) Komunikācijas protokols, kas var būt TCP vai UPD. Ja arguments ir 0, tad protokols tiek izvēlēts, balstoties uz norādīto ligzdas tipu.

Šajā programma ligzdas tips ir nokonfigurēts, lai strādātu ar standarta parametriem, taču ir iespējami citi ligzdas konfigurāciju veidi.

```
bzero((char *) &serv_addr, sizeof(serv_addr));
```

Funkcija `bzero()` attīra buferi. Pirmais funkcijas arguments ir rādītājs uz buferi, bet otrais – bufera izmērs.

```
portno = atoi(argv[1]);
```

Tiek nokonfigurēts komunikācijas porta numurs. Funkcija `atoi()` tiek izmantota, lai pārveidotu simbolu virkni skaitlī.

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(portno);
serv_addr.sin_addr.s_addr = INADDR_ANY;
```

Tiek konfigurēti servera adreses struktūras mainīgie. `sin_family` satur servera adrešu kopu. `sin_port` satur porta numuru, kas tika pārveidots ar funkcijas `htons()` palīdzību. `sin_addr.s_addr` satur servera adresi, kas atbilst datora adresei un tiek vienmēr norādīts ar konstanti `INADDR_ANY`.

```
if(bind(sockfd,(struct sockaddr*)&serv_addr,
sizeof(serv_addr)) < 0)
error("ERROR on binding");
```

Šis IF bloks saista ligzdu ar adresi. Ja notiek kļūda, tad tiek izvadīts atbilstošais paziņojums.

```
listen(sockfd,5);
```

Funkcija `listen()` ļauj procesam klausīties ligzdu un uztvert savienojumus. Pirmais funkcijas arguments ir izveidotā ligzda, un otrais arguments ir gaidīšanas rindas garums, kas parasti ir 5 (maksimālais skaits, kuru atļauj sistēma).

```
clilen = sizeof(cli_addr);
newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
if (newsockfd < 0)
  error("ERROR on accept");
```

Tiek nolasītas klienta adreses garums un funkcija `accept()` bloķē procesu, kamēr netiek uztverts pieslēgums no klienta.

```
bzero(buffer,256);
n = read(newsockfd,buffer,255);
if (n < 0) error("ERROR reading from socket");
printf("Here is the message: %s", buffer);
```

Komunikācijas buferis tiek attīrīts, izmantojot `bzero()` funkciju. Ar funkciju `read()` saņemtais no klienta ziņojums tiek nolasīts no ligzdas un ievietots buferi. Šī funkcija tiek palaista tikai tad, kad klienta daļā izpildās funkcija `write()` un no komunikācijas ligzdas var nolasīt informāciju.

```
n = write(newsockfd,"I got your message",18);
if (n < 0) error("ERROR writing to socket");
```

Kad savienojums starp serveri un klientu ir izveidots, tad abas puses var izmantot rakstīšanas un lasīšanas funkcijas. Šajā gadījumā serveris nosūta klientam ziņojuma saņemšanas apstiprinājumu.

Klienta daļas kods ir diezgan līdzīgs pēc uzbūves, taču ir dažas atšķirības.

`struct hostent *server` ir struktūra ir norāda uz tipu `hostent`, kas ir definēts iesākumfailā `netdb.h`. Līdzīgi kā servera daļas programma tiek nolasīti parametri un izveidota jauna komunikācijas ligzda. Arī servera struktūras lauki tiek aizpildīti līdzīgā veidā, izņemot to, ka šajā gadijumā `server->h_addr` ir simbolu virkne un tāpēc tiek izmantota funkcija `void bcopy(char *s1, char *s2, int length)`.

```
if (connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0)
  error("ERROR connecting");
```

Funkcija `connect()` tiek izmantota, lai izveidotu savienojumu ar serveri. Gadījumā, ja ar serveri savienoties nevar, tiek izvadīts paziņojums par kļūdu. Funkcijas parametri ir izveidotā ligzda, rādītais uz servera adresi un servera adreses izmērs.

```
printf("Please enter the message: ");
bzero(buffer,256);
fgets(buffer,255,stdin);
n = write(sockfd,buffer,strlen(buffer));
if (n < 0)
error("ERROR writing to socket");
bzero(buffer,256);
n = read(sockfd,buffer,255);
if (n < 0)
error("ERROR reading from socket");
printf("%s",buffer);
return 0;
```

Atšķirībā no servera daļas komunikācijas koda, klients sākumā padod datus (ziņojumu) no bufera uz ligzdu, tad atkārtoti attīra buferi un tad nolasa no ligzdas apstiprinājuma ziņojumu no servera.

Secinājumi

Trešā praktiskā darba gaitā tika izveidots savienojums starp LPC-2478-STK izstrādes plati un datoru, izmantojot Ethernet savienojumu un „Linux networking socket" slāņu pieeju. Izmantotais transporta slāņa protokols ir TCP.

Datora palaistais serveris ir veiksmīgi saņēmis ziņojumu, no LPC LPC-2478-STK izstrādes platē palaista klienta.

Darba rezultāta tika iegūtas papildus zināšanas par UPD un TCP transporta slāņa protokoliem un ligzdām, kā arī tika apskatītas tīkla ligzdu programmēšanas iespējas, izmantojot C programmēšanas valodu un papildus bibliotēkas.

Pielikums

Pirmkodu avots: http://www.linuxhowtos.org/C_C++/socket.htm

Servera daļas pirmkods:

```
#include <stdio.h>       //nepieciešamās standarta C un ligzdu bibliotēkas
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(const char *msg)  //savienojuma kļūdu apstrādes funkcija
{
   perror(msg);
   exit(1);
}

int main(int argc, char *argv[])
{
    int sockfd, newsockfd, portno; //ligzdas un porta mainigie
    socklen_t clilen;
    char buffer[256]; //buferis
    struct sockaddr_in serv_addr, cli_addr; //adresu strukturas
    int n;
    if (argc < 2) { //ja palaišanas laika nav noradits savienojuma ports
       fprintf(stderr,"ERROR, no port provided\n");
       exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0); //ligzdas definesana un atversana
    if (sockfd < 0)  //ja ligzdas atversanas laika notiek kluda
      error("ERROR opening socket");
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]); //porta nolasisana un definesana
    serv_addr.sin_family = AF_INET; //serv_addr struktūras konfiguresana
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
          sizeof(serv_addr)) < 0)
          error("ERROR on binding");
    listen(sockfd,5); //serveris klausas ligzdu
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd, //datu pakesu pienemsana no klienta
             (struct sockaddr *) &cli_addr,
             &clilen);
    if (newsockfd < 0) //ja pieņemot notiek kludas
       error("ERROR on accept");
    bzero(buffer,256);
```

```
    n = read(newsockfd,buffer,255); //nolasisana no ligzdas
    if (n < 0) error("ERROR reading from socket"); //ja nolasot notiek kluda
    printf("Here is the message: %s\n",buffer);
    n = write(newsockfd,"I got your message",18); //suta atbildi klientam
    if (n < 0) error("ERROR writing to socket"); //ja notiek kluda
    close(newsockfd); //sanemsanas ligzdas aizversana
    close(sockfd); //savienojuma aizversana
    return 0;
}
```

Klienta daļas pirmkods:

```
#include <stdio.h>       //nepieciešamās standarta C un ligzdu bibliotēkas
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg)  //savienojuma kļūdu apstrādes funkcija
{
   perror(msg);
   exit(0);
}

int main(int argc, char *argv[])
{
   int sockfd, portno, n; //ligzdas un porta mainigie
   struct sockaddr_in serv_addr; //servera adreses struktūra
   struct hostent *server;

   char buffer[256]; //buferis
   if (argc < 3) { //ja palaišanas laika nav pareizi ievaditi parametri
      fprintf(stderr,"usage %s hostname port\n", argv[0]);
      exit(0);
   }
   portno = atoi(argv[2]); //porta atversana
   sockfd = socket(AF_INET, SOCK_STREAM, 0); //ligzdas definesana un atversana
   if (sockfd < 0)  //ja atverot ligzdu notiek kluda
      error("ERROR opening socket");
   server = gethostbyname(argv[1]); //noradītā servera adreses nolasisana
   if (server == NULL) { //ja noraditais serveris nav atrasts
      fprintf(stderr,"ERROR, no such host\n");
      exit(0);
   }
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sin_family = AF_INET; //serv_addr struktūras konfiguresana
   bcopy((char *)server->h_addr,
```

```
        (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin); //nolasa ievadito ziņojumu
    n = write(sockfd,buffer,strlen(buffer)); //raksta uz ligzdu
    if (n < 0)
         error("ERROR writing to socket"); //izvada, ja rakstot notiek kluda
    bzero(buffer,256);
    n = read(sockfd,buffer,255); //nolasa atbildi no ligzdas
    if (n < 0)
         error("ERROR reading from socket"); //izvada, ja lasot notiek kluda
    printf("%s\n",buffer); //izvada atbildes pazinojumu
    close(sockfd); //aizver savienojumu
    return 0;
}
```