3.Teorētiska daļa

Laboratorijas darba uzdevums: realizēt sakarus starp Olimex izstrādes plati un host datoru izmantojot Ethernet savienojumu. Savienojumam izmantot „Linux networking socket" slāņu pieeju. Kā transporta protokolu izmantot UDP.

3.1Ligzda

Tīkla ligzda(angl. Socket) ir starpprocesu komunikācijas(angl. IPC – Inter Process Communication) plūsmas galapunkts datoru tīklā. Ligzdas lietojumprogrammas saskarni parasti nodrošina operētājsistēma, kas atļauj datorprogrammām izmantot tīkla ligzda. Ligzdas lietojumprogrammas balstās uz Bērklija(angl. Berkeley) ligzdu standarta. Ligzdas adrese ir IP-adreses un porta kombinācija. Pamatojoties uz šo adresi, interneta ligzda piegādā ienākošas datu paketes atbilstošam lietojuma procesam vai pavedienam. [1]

3.1.1Bezsavienojuma un savienojumorientēts protokols[2]

- Interneta Protokola terminoloģijā, pamata datu pārsūtīšanas vienība ir datagramma(angl. Datagram). Parasti tā sastāv no galvenes aiz kā seko dati. Datagrammas ligzda ir bezsavienojuma(angl. Connectionless).
- lietotāja datagrammu protokols(angl. UDP – User Datagram Protocol)
    1. bezsavienojuma,
    2. viena ligzda var saņemt un sūtīt paketes no\uz vairākiem datoriem,
    3. lielāka snieguma piegāde,
    4. dažas paketes var tikt pazaudētas vai tikt bojātas.
- pārraides vadības protokols(angl. TCP – Transmission Control Protocol)
    1. savienojumorientēts,
    2. klienta ligzda savienojas ar serveri,
    3. nodrošina divvirzienu kanālu starp klientu un serveru,
    4. pazaudēti dati tiek pārsūtīti,
    5. dati tiek sūtīti secīgi,
    6. dati tiek sūtīti kā bitu plūsma
    7. TCP izmanto plūsmas kontroli

1 http://en.wikipedia.org/wiki/Network_socket

2 http://www.tenouk.com/Module39a.html

- Ir viegli vienam UDP serverim saņemt datus no vairākiem klientiem un atbildēt.
- Ir vieglāk tikt gala ar tīkla problēmām izmantojot TCP.

Apkopojot iepriekšteikto, TCP protokolu pielieto tādos lietojumos, kur ir svarīga pārsūtīto datu robustums, turpretim UDP pielieto tur, kur ir svarīgs pārsūtīšanas ātrums.

3. Praktiska daļa

Iepriekšējā darba ietvaros jau tika nokonfigurēts Ethernet savienojums ar plati, kas noderēs arī šajā darbā. Lai nodemonstrētu lietotāja realizētu Ethernet savienojumu starp plati un datoru izmantojot UDP protokolu, tika izmantots kods no http://www.tenouk.com/ avota(pirmkodu var apskatīt pielikumā).

Vispirms, lai tiktu pie pieejamiem mainīgiem no Interneta Protokola saimes vajag iekļaut standarta bibliotēkas:

- arpa/inet.h – interneta operāciju definēšana, piemēram, inet_network, inet_makeaddr;
- netinet/in.h – IP mainīgie, piemēram, sockaddr_in, in_addr;
- sys/socket.h – IP mainīgie, piemēram, sockaddr;
- sys/types.h – datu tipi;

Lai izveidotu galapunktu komunikācijai serverī vai klientā – jāizmanto funkciju *int socket* ar trim argumentiem(sīkāku informāciju par katru funkciju var iegūt ierakstot terminālī ***man funkcijas_nosaukums***, piemēram, ***man socket***):

- int domain – jābūt uzstādīts kā AF_NET(abr. Address Family),
- int type – norāda kāda tipa ligzda tiks izmantota: SOCK_DGRAM – udp protokols, SOCK_STREAM – tcp,
- int protocol – pieejamos protokolu tipus var apskatīt /etc/protocols. Uzstāda 0, automātiskai izvēlei.

Ja notika kļūda, funkcija atgriež negatīvu skaitli, citādi tiks atgriezts pozitīvs skaitlis(angl. File descriptor – datnes deskriptors), kas tiks izmantots tālāk, lai:

- piesiet vārdu ligzdai serverī – *int bind(int sockfd, struct sockaddr *my_addr, int addrlen),* kur
  - sockfd(abr. Socket File Descriptor),
  - sockaddr(abr. Socket Address) – ir radītājs uz struktūru, kurā glabājas informācija par adresi, proti, IP un portu,
  - addrlen(abr. Address Length) – struktūras izmērs baitos.
- Saņemt paketes - *int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen)*, kur

- *buf – radītājs uz atmiņas bloku, kur tiks saglabāti saņemtie dati,
- len – atmiņas bloka izmērs,
- flags – komunikācijas opcijas, piemēram, MSG_DONTWAIT - ; norādot 0, šī funkcija ir ekvivalenta write() funkcijai,
- *from – radītājs uz struktūru, kas glabās informāciju par sūtītāju,
- *fromlen – glabā saņemtas adreses reālu izmēru.

- Sūtīt paketes - *int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen)*, kur
  - *msg – radītājs uz atmiņas bloka sākumu, kurā glabājas ziņojums,
  - len – ziņojuma atmiņas bloka izmērs,
  - *to – rādītājs uz saņēmēja adreses informāciju.

  Struktūras *sockaddr*, tiek uzstādītas šādas vērtības:

  - sin_family – AF_NET,
  - sin_port – izmantots porta numurs,
  - sin_addr.s_addr – tiek uzstādīta servera IP adrese(automātiskai izvēlei jāuzstāda INADDR_ANY)

Ar funkcijas *memset(&variable, '\0', 8*) palīdzību tiek attīrītas vērtības, kuras nav nepieciešamas dotā lietojumā.

Servera implementācijas pseido-kods izskatās šādi:

Izveidot ligzdu(int socket(...))

Piesiet to pie zināma porta(int bind(...))

Gaidīt pieprasījumu no klienta(int recvfrom(...))

Nosūtīt atbildi klientam(int sendto(...))

aizvērt ligzdu(int close(...))

Klienta implementācijas pseido-kods ir līdzīgs, taču bez porta piesiešanas, jo tas nav vajadzīgs. Tiek izveidota ligzda un nosūtīts pieprasījums. Kad tiek saņemta atbilde no servera, ligzda tiek aizvērta un programma beidz darbību.

Darbības rezultāts:

```
colt@colt-N53SV:/media/DATA/Sketchbook/c/unix_socket$ ./server
Server-socket() sockfd is OK...
Server-bind() is OK...
Server-Waiting and listening...
Server-recvfrom() is OK...
Server-Got packet from 127.0.0.1
Server-Packet is 15 bytes long
Server-Packet contains "MSG FROM CLIENT"
Server-sockfd successfully closed!
colt@colt-N53SV:/media/DATA/Sketchbook/c/unix_socket$
```

Attēls 1: Palaižot serveri tas gaida(Server-Waiting...) un kad tiek palaists klients, apstrādā saņemtos datus

```
colt@colt-N53SV:/media/DATA/Sketchbook/c/unix_socket$ ./client localhost "MSG FR
OM CLIENT"
Client-gethostname() is OK...
Client-socket() sockfd is OK...
Using port: 4950
Client-sendto() is OK...
sent 15 bytes to 127.0.0.1
Client-Waiting and listening...
Client-recvfrom() is OK...
Client-Got packet from 127.0.0.1
Client-Packet is 19 bytes long
Client-Packet contains "ACK MSG FROM SERVER"
Client-sockfd successfully closed!
```

Attēls 2: Palaižot klientu tas nosūta ziņojumu un gaida atbildi(Client-Waitin...), kad tā ir saņemta notiek ziņojuma apstrāde

Secinājumi

Dota darba ietvaros tika izveidota klienta servera programma, kas ļauj komunicēt izmantojot ligzdas(angl. Sockets). Programmas darbība bija notestēta linux vidē host datorā un tā kā mikroprocesorā arī tiek izmantots linux, tad, kā arī bija gaidāms, arī tur programmas veiksmīgi darbojas. Darba rezultāta ir iespējams izveidot no plates serveri, kas spēs apstrādāt pieprasījumus no klientiem, vai arī klientu, kas veiks pieprasījumus serverim. Pietam, plate var būt gan serveris, gan klients palaižot programmas kā atsevišķus procesus, kas ir viens no IPC piemēriem.

Dotajā piemērā tika izmantots UDP komunikācijas protokols, kas ir labākais risinājums, kad ir nepieciešams ātrums, nevis datu drošība. Lai šīs serveris varētu komunicēt ar TCP klientu, vajag atsevišķi realizēt šāda gadījuma apstrādi. To realizēt nav sarežģīti – vajag mainīt ligzdas tipu uz SOCK_STREAM un izmantot dažas papildus funkcijas, kā *int listen(...)* TCP servera pusē, kas klausīsies savienojumus portam un jā tāds būs pieņems to(*int accept(...)*) un *int connect(...)* klienta pusē, kas mēģinās pieslēgties serverim.

Pielikums I Servera pirmkods

```
/*receiverprog.c – a server, datagram sockets*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
/* the port users will be connecting to */
#define MYPORT 4950
#define MAXBUFLEN 500

int main(int argc, char *argv[]){
      int sockfd;
      /* my address information */
      struct sockaddr_in my_addr;
      /* connector's address information */
      struct sockaddr_in their_addr;
      int addr_len, numbytes;
      char buf[MAXBUFLEN];
      if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1){
            perror("Server-socket() sockfd error lol!");
            exit(1);
      }
      printf("Server-socket() sockfd is OK...\n");
      /* host byte order */
      my_addr.sin_family = AF_INET;
      /* short, network byte order */
      my_addr.sin_port = htons(MYPORT);
      /* automatically fill with my IP */
      my_addr.sin_addr.s_addr = INADDR_ANY;
      /* zero the rest of the struct */
      memset(&(my_addr.sin_zero), '\0', 8);
      if(bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1){
            perror("Server-bind() error lol!");
            exit(1);
      }
      printf("Server-bind() is OK...\n");
      addr_len = sizeof(struct sockaddr);
      if((numbytes  =  recvfrom(sockfd,  buf,  MAXBUFLEN-1,  0,  (struct  sockaddr
*)&their_addr, &addr_len)) == -1){
            perror("Server-recvfrom() error lol!");
            /*If something wrong, just exit lol...*/
            exit(1);
      }
      printf("Server-Waiting and listening...\n");
      printf("Server-recvfrom() is OK...\n");
      printf("Server-Got packet from %s\n", inet_ntoa(their_addr.sin_addr));
      printf("Server-Packet is %d bytes long\n", numbytes);
      buf[numbytes] = '\0';
      printf("Server-Packet contains \"%s\"\n", buf);
      if((numbytes  =  sendto(sockfd,  "ACK  MSG  FROM  SERVER",  strlen("ACK  MSG  FROM
```

```
SERVER"), 0, (struct sockaddr *)&their_addr, sizeof(struct sockaddr))) == -1){
                        perror("Server-sendto() error lol!");
                        exit(1);
                }
            printf("Server-sendto() is OK...\n");
            if(close(sockfd) != 0)
                  printf("Server-sockfd closing failed!\n");
            printf("Server-sockfd successfully closed!\n");
            return 0;
      }
```

Pielikums II Klienta pirmkods

```
/*senderprog.c – a client, datagram*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
/* the port users will be connecting to */
#define MYPORT 4950
#define MAXBUFLEN 500
int main(int argc, char *argv[ ]){
      int sockfd, addr_len;
      /* connector's address information */
      struct sockaddr_in their_addr;
      struct hostent *he;
      int numbytes;
      char buf[MAXBUFLEN];
      if (argc != 3){
            fprintf(stderr, "Client-Usage: %s <hostname> <message>\n", argv[0]);
            exit(1);
      }
      /* get the host info */
      if ((he = gethostbyname(argv[1])) == NULL){
            perror("Client-gethostbyname() error lol!");
            exit(1);
      }
      printf("Client-gethostname() is OK...\n");
      if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1){
            perror("Client-socket() error lol!");
            exit(1);
      }
      printf("Client-socket() sockfd is OK...\n");
      /* host byte order */
      their_addr.sin_family = AF_INET;
      /* short, network byte order */
      printf("Using port: %d\n",MYPORT);
      their_addr.sin_port = htons(MYPORT);
      their_addr.sin_addr = *((struct in_addr *)he->h_addr);
      /* zero the rest of the struct */
      memset(&(their_addr.sin_zero), '\0', 8);
      if((numbytes = sendto(sockfd, argv[2], strlen(argv[2]), 0, (struct sockaddr
*)&their_addr, sizeof(struct sockaddr))) == -1){
            perror("Client-sendto() error lol!");
            exit(1);
      }
      printf("Client-sendto() is OK...\n");
      printf("sent %d bytes to %s\n", numbytes, inet_ntoa(their_addr.sin_addr));
      addr_len = sizeof(struct sockaddr);
   if((numbytes = recvfrom(sockfd, buf, MAXBUFLEN-1, 0, (struct sockaddr *)&their_addr,
&addr_len)) == -1){
```

```
            perror("Server-recvfrom() error lol!");
            /*If something wrong, just exit lol...*/
            exit(1);
      }
      printf("Client-Waiting and listening...\n");
      printf("Client-recvfrom() is OK...\n");
      printf("Client-Got packet from %s\n", inet_ntoa(their_addr.sin_addr));
      printf("Client-Packet is %d bytes long\n", numbytes);
      buf[numbytes] = '\0';
      printf("Client-Packet contains \"%s\"\n", buf);
      if (close(sockfd) != 0)
            printf("Client-sockfd closing is failed!\n");
      printf("Client-sockfd successfully closed!\n");
      return 0;
}
```

4. Teorētiska daļa

Starpprocesu komunikācija(angl. Inter Process Communication - IPC) – metožu kopa, kas ļauj realizēt datu apmaiņu starp vairākiem pavedieniem vienā vai vairāku procesu ietvaros. Procesi var darboties uz viena vai vairākos datoros, kas darbojas kopējā tīklā. Šie metodi ir sadalīti grupas pēc uzdevumiem: ziņojumu pādošana, sinhronizācija, atmiņas koplietošana, un procedūras attālai izsaukums(angl. Remote procedure calls).[3]

Ir vairāki iemesli, lai izmantotu IPC un daži no tiem ir:

- informācijas koplietošana,
- skaitļošanas operāciju paātrināšana,
- modularitāte.

Tabula 1: Galvenas starpprocesu komunikācijas metodes

Metode	**Īss apraksts**	**Kur realizējams**
Fails	Ieraksts diskā, kuram var piekļūt no jebkura procesa pēc nosaukuma	Lielāka OS daļa
Signāls	Sistēmas ziņojums sūtīts no viena procesa otram, parasti komandas.	Lielāka OS daļa
Ligzda	Datu straume sūtīta caur tīkla interfeisu	Lielāka OS daļa
Ziņojumu rinda	Līdzīga programmkanālam, taču dati tiek glabāti un saņemti paketē.	Lielāka OS daļa
Programmkanāls	Divvirzienu datu kanāls ar parastu ieejas\izejas interfeisu. Dati tiks nolasīti simbols pēc simbola	Visas POSIX sistēmas, Windows
Definētais programmkanāls	Programmkanāls sistēmas faila veidā, nevis ieejas\izejas interfeisa veidā.	Visas POSIX sistēmas, Windows
Semafors	Datu struktūra, kas sinhronizē pavedienus vai procesus, kas izmanto koplietojamus datus.	Visas POSIX sistēmas, Windows
Koplietojama atmiņa	Vairākiem procesiem ir pieeja vienam un tam pašam atmiņas blokam, ļaujot visiem to mainīt un lasīt.	Visas POSIX sistēmas, Windows
Ziņojumu padošana	Līdzīgi ziņojumu rindai(nekas nav koplietojams)	Java RMI, CORBA, DD,....
Atmiņai piekārtots fails	brīvpiekļuves atmiņai(angl. RAM) piesaistīts fails, kas var tikt modificēts tieši mainot atmiņas adreses. Manto	Visas POSIX sistēmas, Windows

3 http://en.wikipedia.org/wiki/Inter-process_communication

	parasta faila priekšrocības.	

4. Praktiska daļa

Secinājumi