Programarea rețelelor

Generalități

Arhitectura standard pentru aplicatii in retea este **client-server**.

Serverul este un proces care asteapta **clientii** sa il contacteze.

Mod de functionare (server iterativ):

- dupa pornire procesul server intra intr-o stare de asteptare pana la conectarea unui posibil client;
- procesul client se va conecta la server si va face cereri pentru anumite servicii (servicii pe care le ofera serverul);
- clientul poate fi pornit pe acelasi sistem ca si serverul sau pe un altul;
- conversatia intre client si server se termina in momentul in care clientul a obtinut rezultatul dorit de la server;
- dupa inchiderea conexiunii serverul revine la starea de asteptare a potentialilor clienti.

Serverul poate fi de doua tipuri:

- iterativ
  - serveste clientii in mod secvential;
  - timpul de conectare este limitat;
  - clientii sunt serviti chiar de procesul server.

- concurent
  - clientii sunt serviti intr-o maniera concurenta;
  - timpii de procesare a cererilor de la clienti sunt necunoscuti;
  - procesul server foloseste una sau ambele metode:
    - va clona procese noi, identice cu procesul original, ce vor servi clientii;
    - va folosi fire de executie pentru tratarea in paralel a clientilor.

Socket-uri

Exista doua metode de comunicare intre client si server: socket-uri Berkeley si TLI (System V Transport Layer Protocol).

Socket-urile au aparut pentru prima data in jurul anului 1982 in BSD 4.1 si sunt cele mai folosite. Ele sunt create explicit, utilizate si puse in functiune de catre aplicatii.

Exista doua tipuri de servicii de transport pentru socket:

- **legaturi orientate pe conexiune**
  - se mai numesc si sigure, ele garantand livrarea datelor, asigurand totodata si integritatea lor.
- **legaturi fara conexiune (nesigure)**
  - se mai numesc si nesigure, ele negarantand livrarea datelor.

In continuare se vor descrie apelurile de functii elementare necesare comunicarii intre client si server.

Functia socket()

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

int socket (int *family*, int *type*, int *protocol*);

unde:

- *family* este familia de protocoale de transport. Poate fi una dintre:

AF_UNIX	Unix internal protocols
AF_INET	Internet protocols
AF_NS	Xerox NS protocols
AF_IMPLINK	IMP link layer
AF_IMPLINK	DEC DNA protocols

AF_ este o abreviere de la “adress family”. Se mai poate folosi si PF_ care este o abreviere pentru “protocol family”. Cele doua notatii sunt echivalente.

Socket-uri

- *type* este unul dintre:

SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_RAW	raw socket
SOCK_SEQPACKET	sequenced packet socket

Nu toate combinatiile de *family* si *type* sunt valide.

	AF_UNIX	AF_INET
SOCK_STREAM	Da	TCP
SOCK_DGRAM	Da	UDP
SOCK_RAW		IP

- *protocol* se initializeaza cu 0.

Functia returneaza un descriptor de socket (int), similar cu descriptorul de fisier in cazul functiei fopen(). In caz de eroare se returneaza -1.

Functia bind()

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

int bind (int *sockfd*, struct sockaddr **myaddr*, int *addrlen*);

unde:

- *sockfd* este descriptorul de socket returnat de functia soket();
- **myaddr* este un pointer catre o structura *sockaddr_in* ce pastreaza adresa locala;
- *addrlen* este dimensiunea structurii *myaddr*.

Functia returneaza 0 in caz de succes sau -1 in cazul unei erori, caz in care errno este setat corespunzator.

Structura *sockaddr_in* are forma:

```
 struct sockaddr_in {
        short       syn_family;         /* AF_INET */
        u_short sin_port                /* 16-bit port number */
                                        /* network byte order */
        struct in_addr sin_addr         /* 32-bit netid/hostid */
                                        /* network byte order */
        char sin_zero[8] /* unused */
};
```

iar structura *in_addr* forma:

```
struct in_addr {
        u_long s_addr;                  /* 32-bit netid/hostid */
                                        /* network byte order */
};
```

Socket-uri

Deoarece exista multe arhitecturi de echipamente de calcul ce pot stoca numere intregi pe 16/32 biti in diferite moduri de ordonare in memorie, sunt necesare rutine de conversie a acestora intr-un format comun NBO – Network Byte Order.

Acestea sunt:

Functie	Actiune
htonl()	converteste formatul gazda 32 bit in nbo
ntohl()	converteste nbo in formatul gazda 32 bit
htons()	converteste formatul gazda 16 bit in nbo
ntos()	converteste nbo in formatul gazda 16 bit

Daca dorim ca procesul server sa asculte pe toate interfetele de retea existente pe sistem se poate folosi ca valoare pentru *s_addr* constanta INADDR_ANY.

Exemplu:

```
#define MY_PORT_ID 6666

struct sockaddr_in ssock_addr;

bzero((char *) &ssock_addr, sizeof(ssock_addr));

ssock_addr.sin_family = AF_INET;
ssock_addr.sin_addr.s_addr = htonl(INADDR_ANY);
ssock_addr.sin_port = htons(MY_PORT_ID);
```

Functia listen()

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

int listen (int *sockfd*, *int backlog*);

unde:

- *sockfd* este descriptorul de socket returnat de functia soket();
- *backlog* specifica cate conexiuni pot fi puse intr-o coada de asteptare de catre sistem in timp ce serverul ruleaza apelul sistem accept().

Functia este folosita de serverul cu conexiune orientata pentru a semnala ca doreste sa accepte conexiuni.

Functia accept()

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

int accept (int *sockfd*, struct sockaddr **fromaddr*, int *addrlen*);

unde:
- *sockfd* este descriptorul de socket returnat de functia soket();
- **fromaddr* este un pointer catre o structura ce pastreaza adresa remote (specifica pentru un anumit protocol) – adresa socket-ului ce transmite date;
- *addrlen* este dimensiunea structurii *fromaddr*.

Functia accept() returneaza un nou socket cu aceleasi proprietati ca si *sockfd* prin care se vor transmite date intre client si server.

Functia returneaza -1 pentru in cazul unei erori.

Functia connect()

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

int connect (int *sockfd*, struct sockaddr **toaddr*, int *addrlen*);

unde:
- *sockfd* este descriptorul returnat de functia socket();
- **toaddr* este un pointer catre o structura ce pastreaza adresa remote – adresa serverului;
- *addrlen* este dimensiunea structurii *toaddr*.

Functia returneaza 0 pentru succes sau -1 in cazul unei erori, caz in care errno este setat corespunzator.

Functii de citire din respectiv scriere in socket:

Functiile read() si write()

Apelurile sistem read(), write() sunt apeluri normale ca si in cazul citirii dintr-un fisier.

Functiile send(), sendto(), recv() si recvfrom()

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

int send (int *sockfd*, char **buff*, int *nbytes* int *flags*);
int sendto (int *sockfd*, char **buff*, int *nbytes,* int *flags*, struct sockaddr **to*, int *addrlen*);
int recv (int *sockfd*, char **buff*, int *nbytes* int *flags*);
int recvfrom (int *sockfd*, char **buff*, int *nbytes,* int *flags*, struct sockaddr **from*, int *addrlen*);

unde:
- *sockfd*, *buff* si *nbytes* pentru toate cele patru functii, sunt similari cu parametri functiilor read() si write().

- flags este 0 sau un OR logic intre urmatoarele constante:

MSG_OOB	trimite si receptioneaza date “out-of-band”
MSG_PEEK	lasa apelantul sa vada daca sunt date care sa fie citite fara ca sistemul sa renunte la date dupa apelul lui recv() sau recvfrom()
MSG_DONTROUTE	bypass route (send() si sendto())

Toate functiile returneaza lungimea datelor scrise sau citite sau -1 in caz de eroare.

Functia close()

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

int close (*int sockfd*);

unde:
- *sockfd* este descriptorul de socket returnat de functia soket().

Functia inchide un descriptor de socket deschis de catre socket(). Returneaza 0 in caz de succes sau -1 in caz de eroare.

Exemplu

server.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <errno.h>
#define MY_PORT_ID 6666 /* a number > 5000 */

int main()
{

  int sockid, newsockid, i, j;
  struct sockaddr_in ssock_addr;
  char msg[255];
```

Exemplu


```
if ((sockid = socket (AF_INET, SOCK_STREAM, 0)) < 0)
{
        perror("Eroare creare socket");
        return -1;
}

bzero((char *) &ssock_addr, sizeof(ssock_addr));
ssock_addr.sin_family = AF_INET;
ssock_addr.sin_addr.s_addr = htonl(INADDR_ANY);
ssock_addr.sin_port = htons(MY_PORT_ID);

if (( bind(sockid, (struct sockaddr *) &ssock_addr,
                sizeof(ssock_addr)) < 0))
{
        perror("Eroare de asociere");
        return -1;
}
```

Exemplu


```
 if ( listen(sockid, 5) < 0)
 {
        perror("Eroare la ascultare");
        return -1;
 }
while (1)
 {
        newsockid = accept(sockid, (struct sockaddr *) 0, (int *) 0);
        if (newsockid < 0)
        {
           perror("Eroare acceptare client");
           return -1;
        }

        if ((read(newsockid, &msg, sizeof(msg))) < 0)
        {
           perror("Eroare citire din socket");
           return -1;
        }
```

Exemplu


```
            printf("Clientul spune: %s\n", msg);

            sprintf(msg, "==> mesaj de la server <==");

            write(newsockid, &msg, sizeof(msg));
            close(newsockid);
    }

    close(sockid);

    return 0;
}
```

Exemplu

client.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MY_PORT_ID 6666 /* numar > 5000 */
#define SERV_HOST_ADDR "10.6.14.190"

int main()
{
   int sockid;
   struct sockaddr_in ssock_addr;
   char msg[255];
```

Exemplu


```
if ((sockid = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
        perror("Eroare creare socket");
        return -1;
}

bzero((char *) &ssock_addr, sizeof(ssock_addr));

ssock_addr.sin_family = AF_INET;
ssock_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
ssock_addr.sin_port = htons(MY_PORT_ID);

if (connect(sockid, (struct sockaddr *) &ssock_addr,
            sizeof(ssock_addr)) < 0)
{
        perror("Eroare conectare la server");
        return -1;
}
```

Exemplu


```
    sprintf(msg, "==> mesaj de la client <==");

    if ( (write(sockid, &msg, sizeof(msg))) < 0)
    {
            perror(“Eroare scriere in socket");
            return -1;
    }
   bzero((char *) &msg, sizeof(msg));
    if ( (read(sockid, &msg, sizeof(msg))) < 0)
    {
            perror("Eroare citire din socket");
            return -1;
    }
    printf("Serverul spune: %s\n", msg);

    close(sockid);

    return 0;
}
```