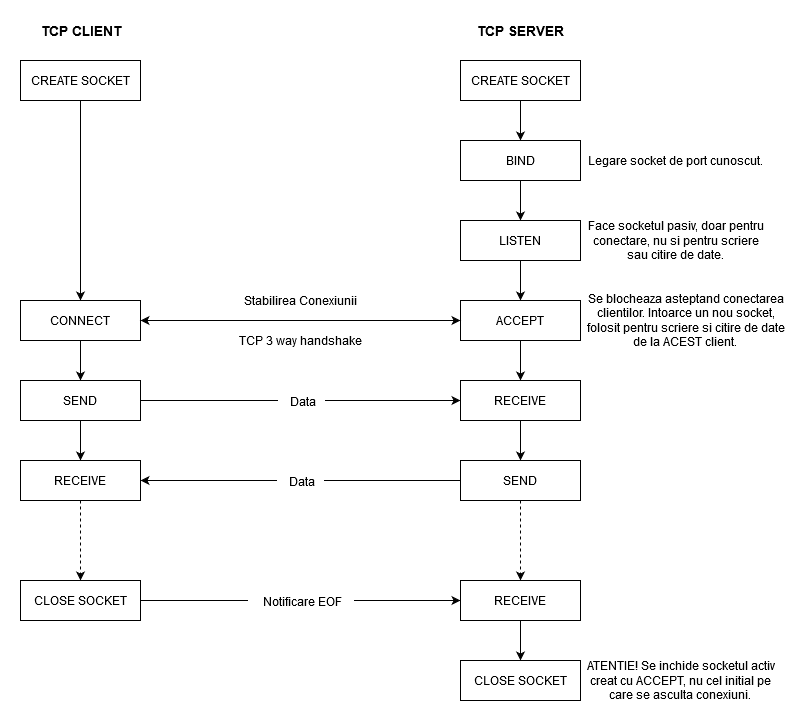
Laborator 7 - TCP



| **Server** | **Client** |
| --- | --- |
| */\* int socket(int domain, int type, int protocol); \*/*  int sockfd = socket(AF\_INET, SOCK\_STREAM, 0)  if (sockfd < 0) { /\* trateaza eroare \*/ } | |
| struct sockaddr\_in serv\_addr;  serv\_addr.sin\_family = AF\_INET;  serv\_addr.sin\_port = htons(1234); | |
| serv\_addr.sin\_addr.s\_addr = INADDR\_ANY; | inet\_aton(“127.0.0.1”, &serv\_addr.sin\_addr) |
| */\*int bind(int sockfd,*  *struct sockaddr \*my\_addr,*  *int addrlen)\*/*  int rs = bind(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr));  if (rs < 0) { /\* trateaza eroare \*/ } |  |
| */\* int listen(int sockfd, int backlog); \*/*  int rs = listen(sockfd, 1));  if (rs < 0) { /\* trateaza eroare \*/ } |  |
| */\* int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen); \*/*  rs < 0  struct sockaddr\_in client\_addr;  socklen\_t socket\_len = sizeof(struct sockaddr\_in);  int connfd = accept(sockfd, (struct sockaddr \*)&client\_addr&client\_addr, &socket\_len);  if (connfd < 0) { /\* trateaza eroare \*/ } | /\* int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen); \*/  int rs = connect(sockfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));  if (rs < 0} { /\* trateaza eroare \*/ } |
| */\* ssize\_t recv(int connfd, void \*buf, size\_t len, int flags); \*/*  recv(connfd, ...) | */\* ssize\_t send(int connfd, const void \*buf, size\_t len, int flags); \*/*  send(sockfd, ) |
| send(connfd, ...) | recv(sockfd, ) |
| /\* int close(ind fd); *#include <unistd.h>* | |

La TCP, send/recv adauga sau citesc date intr-un flux.

Ce inseamna?

1. Cu un recv putem citi date de la mai multe send-uri (senderul poate face 5 senduri, iar recv sa citeasca tot dintr-odata)
2. Recv se deblocheaza cand “ceva date sunt disponibile”

Solutii:

1. incadrarea mesajelor
2. delimitare a mesajelor

Exercitii:

1. Echo server

- 2 terminale

- citim ceva in client, se trimite catre server, serverul trimite inapoi catre client

- fiecare afiseaza ce a trimis

- citim in client -> vedem in server ce s-a trimis -> vedem in client (echoul)

2.

SERVER

// Obtine un socket

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0); // cer un socket pt stream-uri

// Legarea socketului la o adresa cunoscuta

struct sockaddr\_in serv\_addr;

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(1234);

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

bind(listenfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr))

listen(sockfd, 5);

// Acept conexiunea de la cei 2 clienti

connfd1 = accept(listenfd, (struct sockaddr \*) &client\_addr1, &socket\_len);

DIE(connfd1 < 0, "Couldn't accept connection!\n");

printf("Got new connection from %s, port %d, socket %d!\n", inet\_ntoa(client\_addr1.sin\_addr), ntohs(client\_addr1.sin\_port), connfd1);

connfd2 = accept(listenfd, (struct sockaddr \*) &client\_addr2, &socket\_len);

DIE(connfd2 < 0, "Couldn't accept connection!\n");

printf("Got new connection from %s, port %d, socket %d!\n", inet\_ntoa(client\_addr2.sin\_addr), ntohs(client\_addr2.sin\_port), connfd2);

// Trimite fiecarui client un numar pentru a sti care dintre el trebuie sa comunice primul

send(connfd1, "1", strlen("1") + 1, 0);

send(connfd2, "2", strlen("2") + 1, 0);

while (1) {

// primesc de la client1

// trimit catre client 2

// primesc de la client 2

// trimit catre client 1

}

CLIENT

// Obtine un socket

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0); // cer un socket pt stream-uri

// Legarea socketului la o adresa cunoscuta

struct sockaddr\_in serv\_addr;

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(1234);

inet\_aton("127.0.0.1", &serv\_addr.sin\_addr);

connect(sockfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr))

char buff[BUFLEN], client\_nr[BUFLEN];

recv(sockfd, client\_nr, BUFLEN, 0); // ce client sunt: 1 - trimit sau 2 - primesc

if (client\_nr[0] == '1') {

// citesc de la tastatura

// trimit catre server

client\_nr[0] = ‘2’;

} else {

// ordine inversa

// primesc de la server

// afisez pe ecran

client\_nr[0] = ‘1’;

}

close(sockfd);

Pentru exercitiul 2:

Relizati implementarea in client.c intr-o functie numita run\_client2.

Apelati functia run\_client2 in main in loc de apelul lui run\_client.

void run\_client2(int sockfd) {

char client\_nr[2] = {0};

memset(client\_nr, 0, 1);

recv(sockfd, client\_nr, BUFLEN, 0);

printf("Client received number: %s\n", client\_nr);

while (1) {

if (client\_nr[0] == '1') {

if (read(STDIN\_FILENO, buf, BUFLEN - 1) > 0 && !isspace(buf[0])) {

byte\_count = strlen(buf) + 1;

bytes\_remaining = byte\_count;

// TODO a: Trimiteti lungimea mesajului

int rc = send(sockfd, &byte\_count, sizeof(int), 0);

// TODO b: Trimiteti mesajul propriu-zis

while (bytes\_remaining > 0) {

...

}

memset(buf, 0, BUFLEN);

}

client\_nr[0] = '2';

} else if (client\_nr[0] == '2') {

// TODO a: Receptionati lungimea mesajului

int rc = recv(sockfd, &byte\_count, sizeof(int), 0);

// TODO b: Receptionati mesajul propriu-zis

while (bytes\_remaining > 0) {

...

}

fprintf(stderr, "Received from the other client: %s\n", buf);

client\_nr[0] = '1';

}

}

}

Structs:

*#include <netinet/net.h>*

struct sockaddr\_in {

unsigned short sin\_family; // familia de protocoale pe care le folosim

// ex: AF\_INET (IPv4), AF\_INET6 (IPv6)

unsigned short int sin\_port; // portul (in network byte order)

struct in\_addr sin\_addr; // adresa ip pe care ascult dupa conexiuni

// (in network byte order)

}

struct in\_addr {

uint32\_t s\_addr; // adresa IPv4 pe 32 de biti in network byte order

}

*#include <sys/socket.h>*

/\* tip de date generic pentru a retine info despre adresa socketilor \*/

struct sockaddr {

unsigned char sa\_family;

char sa\_data[14];

}