


## Laboratorul 07 - Protocolul de transport TCP

Responsabili: Valeriu Stanciu, Silviu Pantelimon, Radu-Ioan Ciobanu

### Obiective

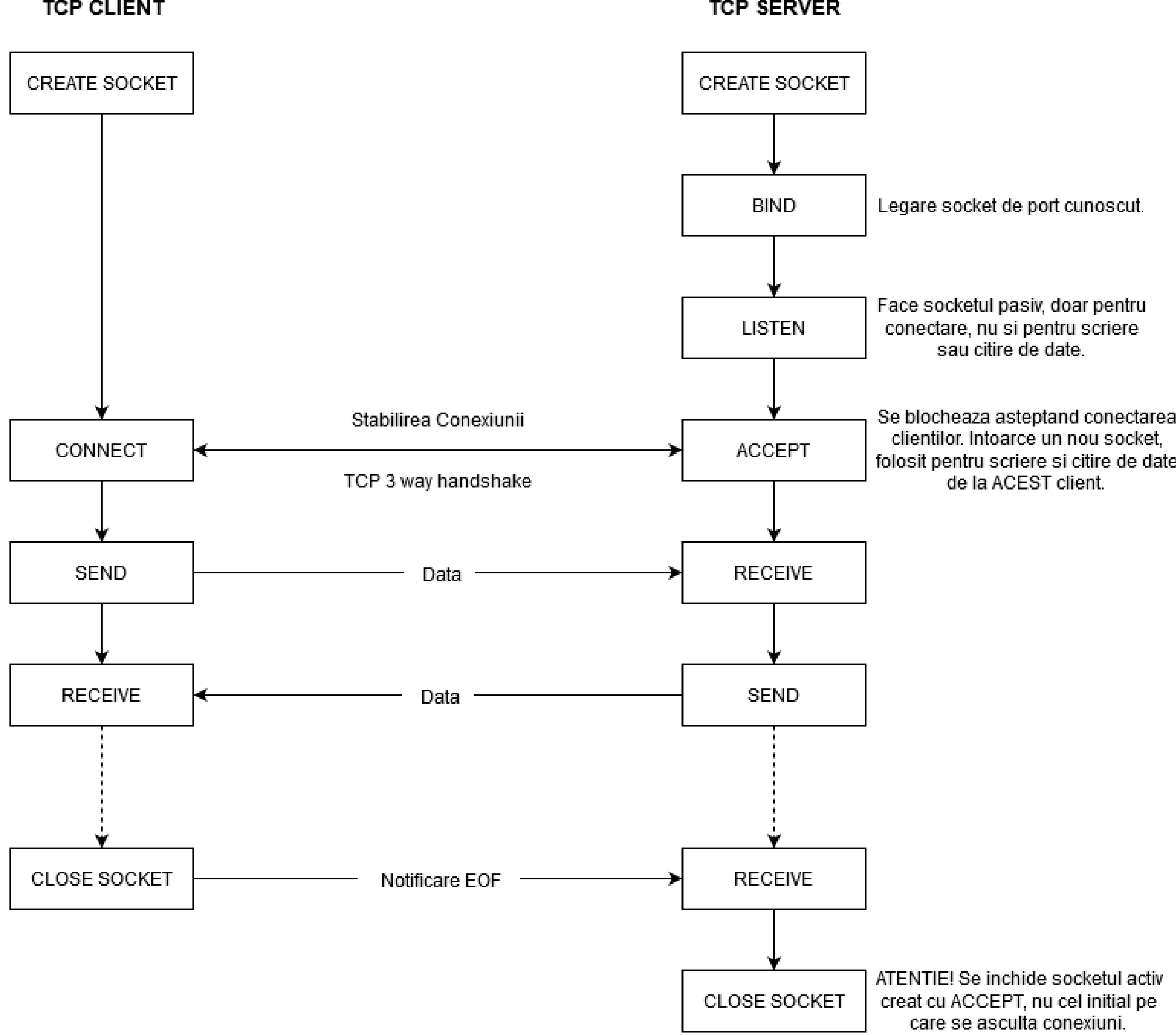
Scopul acestui laborator este ca studenții să se familiarizeze cu ceea ce înseamna protocolul de transport TCP, respectiv să fie capabili să implementeze o aplicație ce utilizează socket-ii TCP.

### Noțiuni teoretice

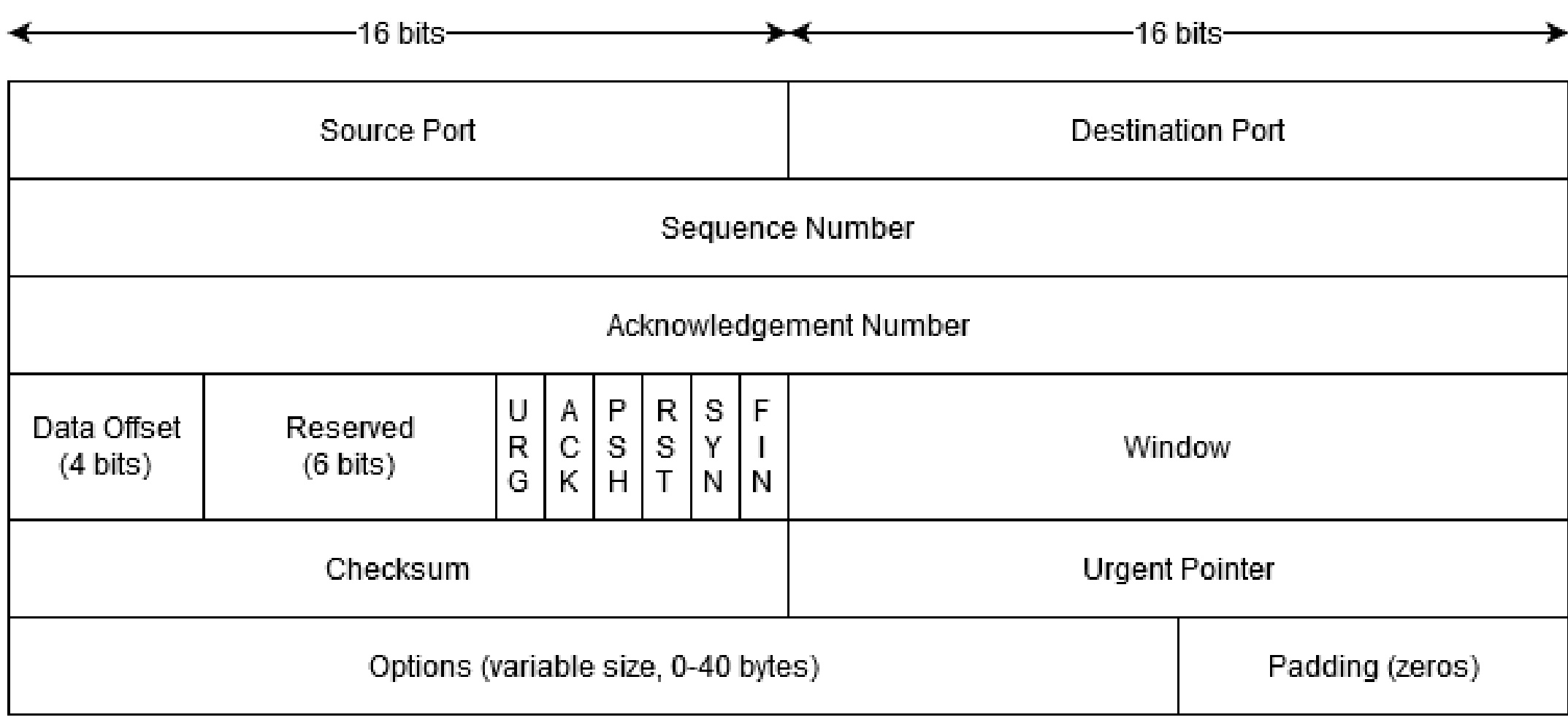
TCP (Transport Control Protocol) este un protocol ce furnizează transmisie garantată (cât timp există conexiune), în ordine și o singură dată, a octeților de la transmitător la receptor. Acest protocol asigură stabilirea unei conexiuni între cele două calculatoare pe parcursul comunicației, și este descris în  RFC 793. Protocolul TCP are următoarele proprietăți:

- stabilirea unei conexiuni între client și server; serverul va aștepta apeluri de conexiune din partea clienților
- garantarea ordinii primirii mesajelor și prevenirea pierderii pachetelor
- controlul congestiei (fereastră glisantă)
- overhead mai mare în comparație cu UDP (are un header de 20 Bytes, spre deosebire de UDP, care are doar 8 Bytes).

### Folosire tipică a API-ului de socket în comunicația TCP



### Header TCP



Explicații header:

- portul sursă** este ales random de către mașina sursă a pachetului, dintre porturile libere existente pe acea mașină
- portul destinație** este portul pe care mașina destinație poate recepționa pachete
- checksum** este valoarea sumei de control pentru un pachet TCP

### API

#### socket()


Pentru obținerea descriptorului de fișier, se folosește, ca și în cazul UDP, funcția  `socket()`:

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

int socket(int domain, int type, int protocol);
```

Câmpurile *domain* și *protocol* au aceleași valori ca și în cazul UDP (*AF\_INET* / *PF\_INET*, respectiv 0). În cazul TCP, pentru câmpul *type* se utilizează valoarea *SOCK\_STREAM*. Diferența notabilă dintre TCP și UDP este în modul de utilizare de către server a descriptorului întors de *socket()*. În cazul UDP, acest socket este implicit activ, adică se fac trimiteri și recepționări de date pe el. În cazul TCP, acest socket va fi folosit numai în mod pasiv, pentru stabilirea conexiunii. Nu se vor trimite sau recepționa date ale aplicației prin intermediul său. Va exista un alt socket (diferit pentru fiecare client) care va fi folosit pentru transmisie.

#### bind()

Odată ce am obținut un socket, trebuie să îi asociem un port pe mașina locală (acest lucru este uzual în cazul în care se dorește așteptarea de către server de conexiuni pe un anumit port). Utilizarea este exact ca în cazul UDP, folosind funcția  `bind()`:

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

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

#### connect()


După ce am creat socketul, clientul trebuie să se conecteze la server, folosind funcția  `connect()`:

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

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Argumentul *sockfd* este un descriptor de fișier obținut în urma apelului *socket()*, *addr* conține portul și adresa IP ale serverului, iar *addrlen* este dimensiunea celui de-al doilea parametru. Ca și în cazul celorlalte funcții, rezultatul este -1 în caz de eroare, iar în caz de succes 0.

#### listen()


Comunicația prin conexiune stabilă este asimetrică. Mai precis, unul din cele două procese implicate joacă rol de server, iar celălalt joacă rol de client. Cu alte cuvinte, serverul trebuie să îi asocieze socketului propriu o adresă pe care oricare client trebuie să o cunoască, și apoi să "asculte" pe acel socket cererile ce provin de la clienți. Mai mult decât atât, în timp ce serverul este ocupat cu tratarea unei cereri, există posibilitatea de a întârzia cererile ce provin de la alți clienți, prin plasarea lor într-o coadă de așteptare. Setarea unui socket pentru a fi pasiv se face prin intermediul funcției neblocaante  `listen()`:

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

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

Argumentul *sockfd* reprezintă descriptorul de fișier obținut în urma apelului *socket()*, iar *backlog* indică numărul de conexiuni acceptate în coada de așteptare. Conexiunile care se fac de către clienți vor aștepta în aceasta coadă până când se face *accept()*, și nu pot fi mai mult de *backlog* conexiuni în așteptare. Apelul *listen()* întoarce 0 în caz de succes și -1 în caz de eroare.

#### accept()


Ce se întâmplă în momentul în care un client încearcă să apeleze *connect()* către o mașină și un port pe care s-a făcut în prealabil *listen()*? Conexiunea va fi pusă în coada de așteptare până în momentul în care se face un apel de  `accept()` de către server. Acest apel întoarce un nou socket care va fi folosit pentru această conexiune:

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

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Argumentul *sockfd* reprezintă socketul pe care s-a făcut *listen()* (deci cel întors de apelul *socket()*). Funcția *accept()* întoarce un nou socket, care va fi folosit pentru operații *send()* / *recv()*. *addr* reprezintă un pointer spre o structură de tip *struct sockaddr* în care se va afla informația despre conexiunea făcută (ce mașină de pe ce port a inițiat conexiunea). Noul socket obținut prin apelul *accept()* va fi folosit în continuare pentru operațiile de transmisie și recepție de date.

#### send() / recv()

Aceste două funcții se folosesc pentru a transmite date prin socket-ii de tip stream sau socket-ii datagramă conectați. Sintaxa pentru trimitere și primire este asemănătoare. Pentru trimitere, se folosește funcția  `send()`:

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

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

Argumentul *sockfd* este socketul căruia se dorește să se trimită date (fie este returnat de apelul *socket()*, fie de apelul *accept()*). Argumentul *buf* este un pointer către adresa de memorie unde se găsesc datele ce se doresc a fi trimise, iar argumentul *len* reprezintă numărul de octeți din memorie începând de la adresa respectivă ce se vor trimite. Funcția *send()* întoarce numărul de octeți efectiv trimiși (acesta poate fi mai mic decât numărul care s-a precizat că se dorește a fi trimis, adică *len*). În caz de eroare, funcția returnează -1, setându-se corespunzător variabila globală *errno*.

Pentru recepție de date, se folosește funcția  `recv()`:

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


ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

În cadrul funcției *recv()*, argumentul *sockfd* reprezintă socketul de unde se citesc datele, *buf* reprezintă un pointer către o adresă din memorie unde se vor scrie octeții citiți, iar *len* reprezintă numărul maxim de octeți ce se vor citi. Funcția *recv()* întoarce numărul de octeți efectiv citiți în *buf* sau -1 în caz de eroare.

Observații:

- recv()* poate întoarce și 0, acest lucru însemnând că entitatea cu care se comunică a închis conexiunea
- pentru scrierea/citirea în/din socket-ii TCP, se pot folosi cu succes și funcțiile *write()* și *read()* (foarte asemănătoare cu *send()* și *recv()*, mai puțin câmpul *flags*, pe care oricum îl setăm pe 0).

#### shutdown()

Pentru întreruperea comunicației pe un socket (pe o parte sau două din comunicația full-duplex), se poate utiliza funcția  `shutdown()`:

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

int shutdown(int sockfd, int how);
```

Al doilea parametru reprezintă partea de comunicație care se întrerupe (citire, scriere, sau ambele).

#### close()

Pentru a închide un socket TCP, se folosește funcția de închidere a unui descriptor de fișier din Unix,  `close()`:

```
#include <unistd.h>

int close(int fd);
```

Atenție, în cazul TCP avem de-a face cu mai mulți socket-ii: unul întors de *socket()* (folosit pentru stabilirea conexiunii cu clienții) și unul sau mai multi socket-ii întorși de *accept()* (câte unul pentru fiecare client conectat), folosiți pentru comunicația cu respectivul client. În cazul în care clientul s-a deconectat (*recv()* întoarce 0), vrem să închidem socketul corespunzător creat cu *accept()*, nu pe cel creat cu *socket()*.

### Exerciții

Pentru implementarea cerințelor, vom porni de la  acest schelet de cod.

- Scrieți o aplicație client-server TCP în care serverul se va comporta ca ecoul clientului (echo server). Într-o buclă, clientul citește un string de la tastatură, îl trimite serverului, așteaptă răspuns de la server și îl afișează. Serverul trimite înapoi clientului același lucru pe care îl primește de la el. Atât serverul cât și clientul primesc la început adresa și portul serverului.
- Completați codul serverului de mai sus astfel încât să funcționeze cu 2 clienți (ambele apeluri de *accept()* trebuie făcute înainte de primul *send()* sau *recv()*). Serverul va intermedia un fel de chat între cei doi clienți: va primi ceva de la un client și va trimite celuilalt, și reciproc. Trebuie avută atenție la ordinea operațiilor (scriere și citire de pe socket) atunci când rulați clienții (în laboratorul viitor, vom folosi în server un mecanism de multiplexare care va elimina acest inconvenient; clienții nu vor mai trebui să scrie și să citească de pe socket într-o anumită ordine).

#### Cursuri

- Cursul 01.
- Cursul 02.
- Cursul 03.
- Cursul 04.
- Cursul 05.
- Cursul 06.
- Cursul 07.
- Cursul 08.
- Cursul 09.
- Cursul 10.
- Cursul 11.
- Cursul 12.

#### Laboratoare

- Laboratorul 01 - Notiuni pregatitoare pentru laboratorul de PC
- Laboratorul 02 - Folosirea unei legaturi de date pentru transmiterea unui fisier
- Laboratorul 03 - Implementarea unui protocol cu fereastra glisanta. Suma de control
- Laboratorul 04 - Forwarding
- Laboratorul 05 - ICMP
- Laboratorul 06 - Socket-ii UDP
- Laboratorul 07 - Protocolul de transport TCP
- Laboratorul 08 - TCP și multiplexare I/O
- Laboratorul 09 - Protocolul DNS
- Laboratorul 10 - Protocolul HTTP
- Laboratorul 11 - E-mail
- Laboratorul 12 - Protocoale de securitate. OpenSSL CLI tools
- Laboratorul 13 - Protocoale de securitate. utilizarea programatica

#### Resurse

- Mașina virtuală

#### Table of Contents

- Laboratorul 07 - Protocolul de transport TCP
  - Obiective
  - Noțiuni teoretice
    - Folosire tipică a API-ului de socket în comunicația TCP
    - Header TCP
  - API
    - socket()
    - bind()
    - connect()
    - listen()
    - accept()
    - send() / recv()
    - shutdown()
    - close()
  - Exerciții