

Note de Laborator
Retele de calculatoare

Specializare: Informatica anul 3

Contact:

retelecdsd@gmail.com

<http://www.cdssd.ro>

Comunicatii de
Date si
Sisteme
Distribuite



<http://www.cdssd.ro>

Laborator 10

1. Obiective:

- Comunicatii multicast: Aplicatii Java; Aplicatii Python
- Aplicatii multi-thread: Aplicatii Java; Aplicatii Python

2. Consideratii teoretice (**Partea practica- pag.12; Tema pag. 20**)

Protocoale de Nivel transport al suitelor de protocoale **TCP/IP**, **OSI**, **NetWare's IPX/SPX**, **AppleTalk**, and **Fibre Channel**:

- ATP, AppleTalk Transaction Protocol
- CUDP, Cyclic UDP
- DCCP, Datagram Congestion Control Protocol
- FCP, Fibre Channel Protocol
- IL, IL Protocol
- MPTCP, Multipath TCP
- RDP, Reliable Datagram Protocol
- RUDP, Reliable User Datagram Protocol
- SCTP, Stream Control Transmission Protocol
- SPX, Sequenced Packet Exchange
- SST, Structured Stream Transport
- TCP, Transmission Control Protocol
- UDP, User Datagram Protocol
- UDP Lite
- μ TP, Micro Transport Protocol
- etc

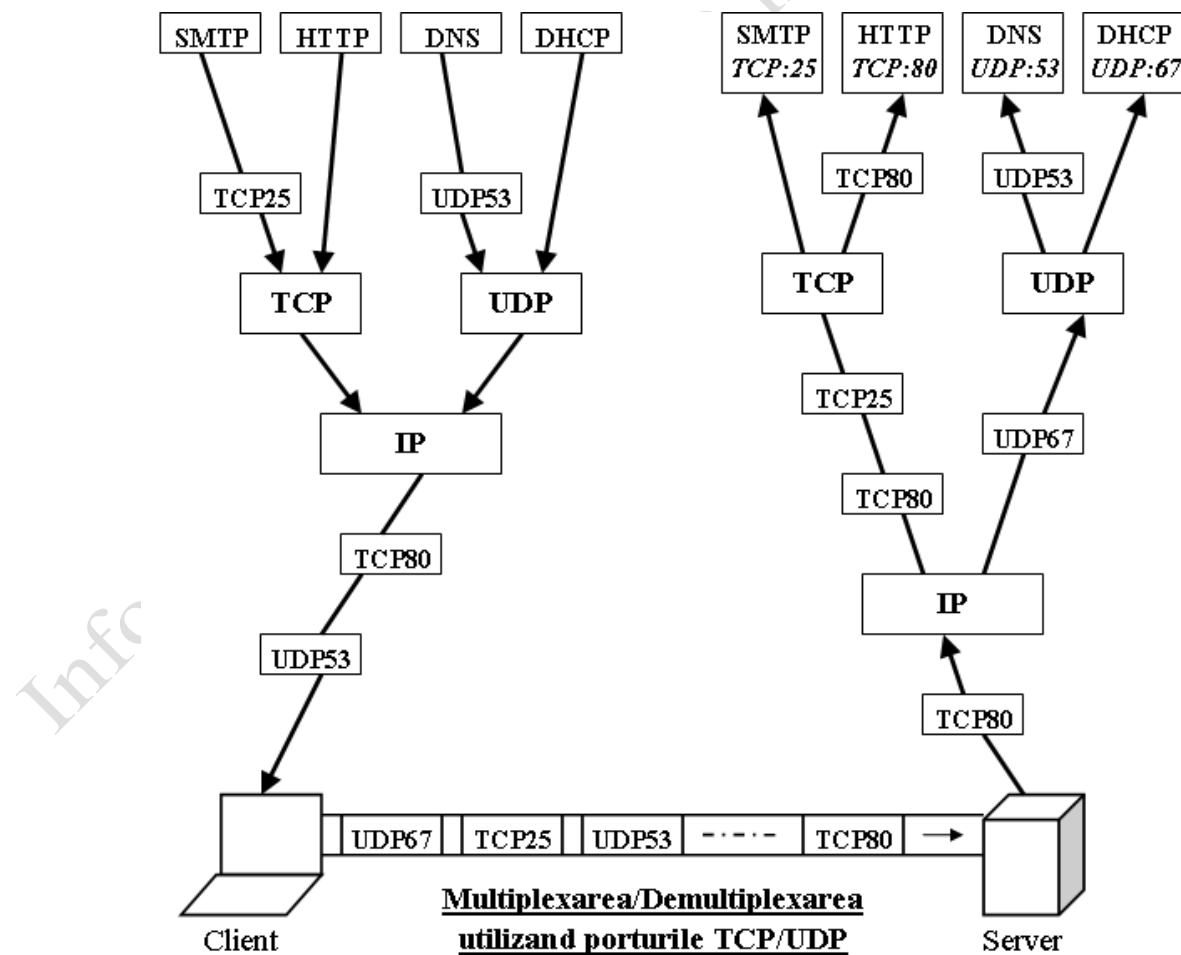
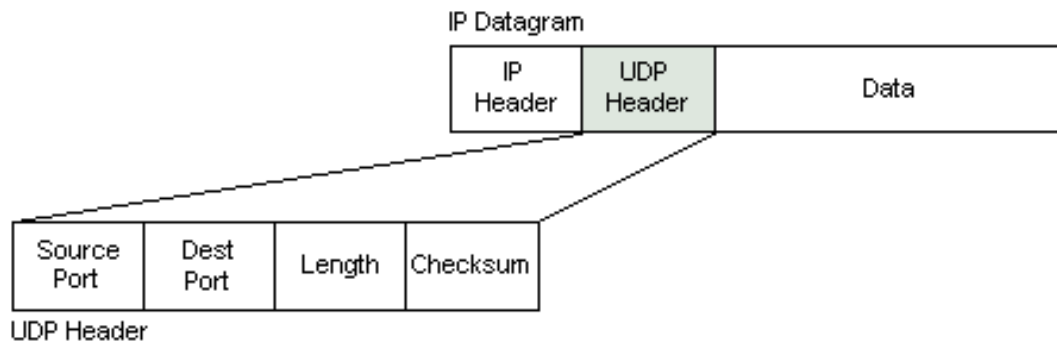
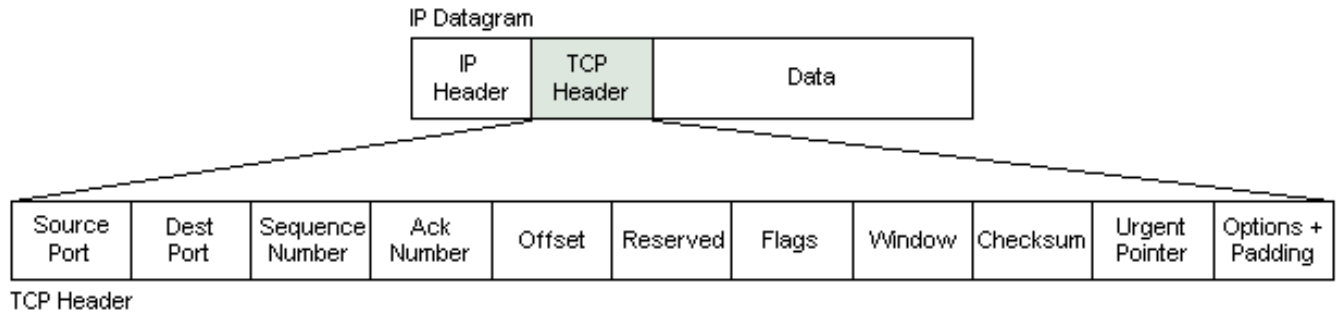
2.1. Suita TCP/IP: TCP (Transmission Control Protocol) si UDP (User Datagram Protocol)

TCP

- ◆ Creaza un circuit virtual intre aplicatiile end-user
 - Orientat conexiune
 - Sigur (reliable)
 - Imparte mesajele in segmente
 - Reasambleaza mesajele la destinatie
 - Retransmite tot ce nu fost primit
 - Reasambleaza segmentele primite din afara

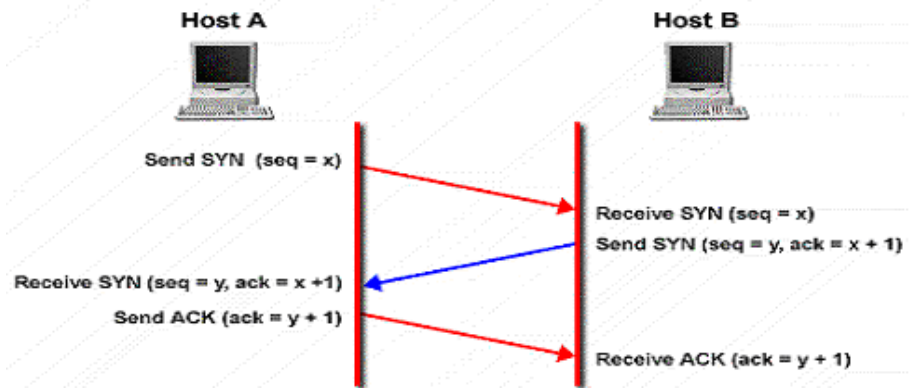
UDP

- ◆ Caracteristici:
 - Fara conexiune
 - Nesigur (unreliable)
 - Transmite mesaje numite datagrame
 - Nu verifica transmiterea mesajelor
 - Nu segmenteaza/reasambleaza mesajele
 - Nu foloseste confirmari
 - Nu face retransmiteri



TCP – Stabilirea conexiunii

TCP Three-Way Handshake/Open Connection



Exemplu:

```

+ Frame 1 (62 bytes on wire, 62 bytes captured)
+ Ethernet II, Src: Xerox_00:00:00 (00:00:01:00:00:00), Dst: fe:ff:20:00:01:00 (fe:ff:20:00:01:00)
+ Internet Protocol, Src: 145.254.160.237 (145.254.160.237), Dst: 65.208.228.223 (65.208.228.223)
+ Transmission Control Protocol, Src Port: tip2 (3372), Dst Port: http (80), Seq: 0, Len: 0
  Source port: tip2 (3372)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 0 (relative sequence number)
  Header length: 28 bytes
+ Flags: 0x02 (SYN)
  0... .... = Congestion window Reduced (CWR): Not set
  .0... .... = ECN-Echo: Not set
  ..0. .... = Urgent: Not set
  ...0 .... = Acknowledgement: Not set
  .... 0... = Push: Not set
  .... .0.. = Reset: Not set
+ .... ..1. = Syn: Set
  .... ...0 = Fin: Not set
  Window size: 8760
+ Checksum: 0xc30c [validation disabled]
+ Options: (8 bytes)

+ Frame 2 (62 bytes on wire, 62 bytes captured)
+ Ethernet II, Src: fe:ff:20:00:01:00 (fe:ff:20:00:01:00), Dst: Xerox_00:00:00 (00:00:01:00:00:00)
+ Internet Protocol, Src: 65.208.228.223 (65.208.228.223), Dst: 145.254.160.237 (145.254.160.237)
+ Transmission Control Protocol, Src Port: http (80), Dst Port: tip2 (3372), Seq: 0, Ack: 1, Len: 0
  Source port: http (80)
  Destination port: tip2 (3372)
  [Stream index: 0]
  Sequence number: 0 (relative sequence number)
  Acknowledgement number: 1 (relative ack number)
  Header length: 28 bytes
+ Flags: 0x12 (SYN, ACK)
  0... .... = Congestion window Reduced (CWR): Not set
  .0... .... = ECN-Echo: Not set
  ..0. .... = Urgent: Not set
  ...1 .... = Acknowledgement: Set
  .... 0... = Push: Not set
  .... .0.. = Reset: Not set
+ .... ..1. = Syn: Set
  .... ...0 = Fin: Not set
  Window size: 5840
+ Checksum: 0x5bdc [validation disabled]
+ Options: (8 bytes)
+ [SEQ/ACK analysis]
  
```

Retele de calculatoare – Informatica anul 3 (2019-2020)

```
Frame 3 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: Xerox_00:00:00 (00:00:01:00:00:00), Dst: fe:ff:20:00:01:00 (fe:ff:20:00:01:00)
Internet Protocol, Src: 145.254.160.237 (145.254.160.237), Dst: 65.208.228.223 (65.208.228.223)
Transmission Control Protocol, Src Port: tip2 (3372), Dst Port: http (80), Seq: 1, Ack: 1, Len: 0
  Source port: tip2 (3372)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 1 (relative sequence number)
  Acknowledgement number: 1 (relative ack number)
  Header length: 20 bytes
  Flags: 0x10 (ACK)
    0... .... = Congestion window Reduced (CWR): Not set
    .0.. .... = ECN-Echo: Not set
    ..0. .... = Urgent: Not set
    ...1 .... = Acknowledgement: Set
    .... 0... = Push: Not set
    .... .0.. = Reset: Not set
    .... ..0. = Syn: Not set
    .... ...0 = Fin: Not set
  Window size: 9660
  Checksum: 0x7964 [validation disabled]
  [SEQ/ACK analysis]
```

2.2. Socket-uri

- Un **socket** (canal de comunicare) este un **punct terminal al unei comunicatii punct-la-punct**, avand un **nume** si o **adresa**. Din perspectiva programatorului, un **socket ascunde detaliile retelei**
- O **adresa socket** pe o retea TCP/IP consta din doua parti: o **adresa IP** si o **adresa (numar) de port**.

Un socket furnizeaza facilitati pentru crearea de fluxuri de intrare/iesire, care permit schimburile de date intre client si server. Atunci cand se stabileste o conexiune, atat clientul cat si serverul vor avea cate un socket, comunicarea efectiva realizandu-se intre socketuri.

Alocarea numarului de porturi este gestionata de IANA pentru a asigura compatibilitate universală pe întregul Internet. Există **trei domenii de numere de porturi**:

- Well-known (Privileged) Port Numbers

0-1023 → system port numbers → utilizate pentru cele mai universale aplicații TCP/IP (standardizate - RFC)

- Registered (user) Port Numbers

1024-49151 → user port numbers → utilizate pentru aplicații neprecizate prin RFC-uri. Pentru a asigura că nu există conflicte, IANA alocă numărul de porturi celor care au creat aplicații server viabile, de regulă accesibile oricărui utilizator.

- Private/Dynamic Port numbers

49152-65535 → nu sunt rezervate și gestionate de IANA. Pot fi utilizate de oricine, fără înregistrare → protocoale private pentru organizații private.

Exista doua forme (nivel transport) ale comunicarii prin sockets:

- orientata spre conexiune
- prin datagrame (neorientat spre conexiune)

Obs: Socket-uri brute (raw sockets – nivel retea http://en.wikipedia.org/wiki/Raw_socket ; http://www.linuxchix.org/content/courses/security/raw_sockets; <http://mixter.void.ru/rawip.html>; <http://www.security-freak.net/raw-sockets/raw-sockets.html>, [http://msdn.microsoft.com/en-us/library/ms740548\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms740548(v=vs.85).aspx), <http://www.savarese.com/software/rocksaw/>)

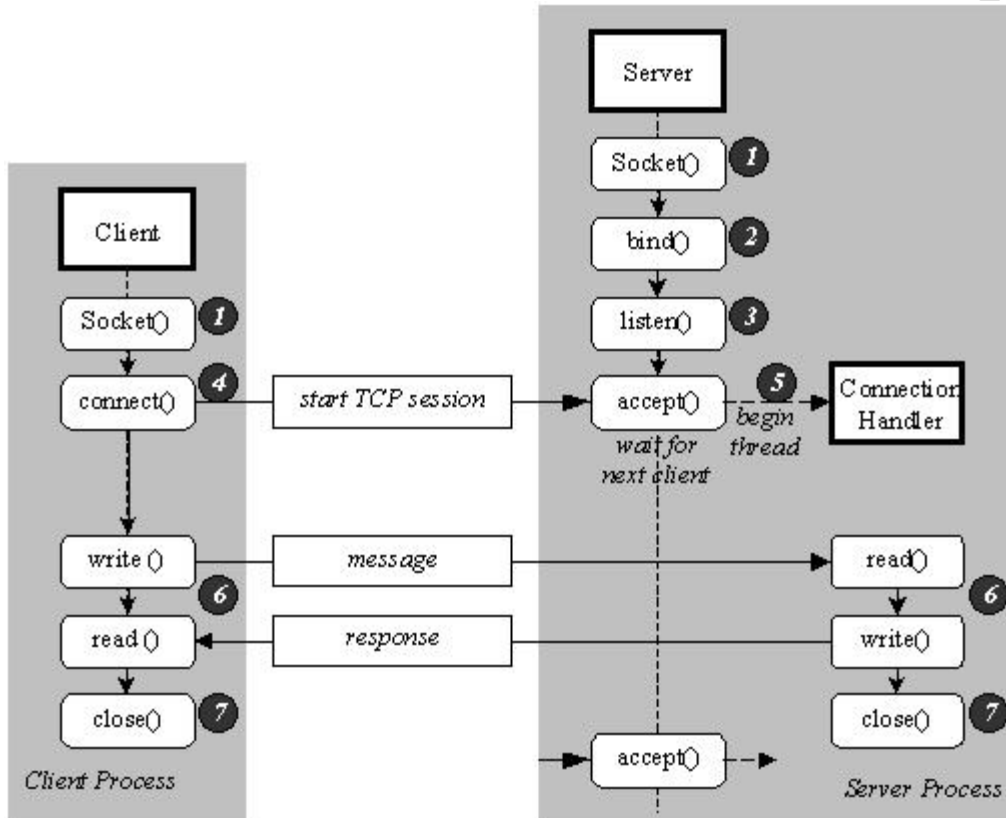
Modelul TCP/IP suporta ambele metode prin implementarea protocolului TCP (Transmission Control Protocol) și a protocolului UDP (User Datagram Protocol).

2.2.1. TCP (Transmission Control Protocol)

Protocolul TCP este orientat spre conexiune aflat pe nivelul transport -asigura servicii de comunicare sigure cu detectarea și corectarea erorilor între două gazde. Stabilirea unei conexiuni se bazează pe adresa IP a mașinii destinație și pe numărul portului pe care aceasta așteaptă cereri de conectare.

Scenariul unei comunicatii bazate pe socket-uri

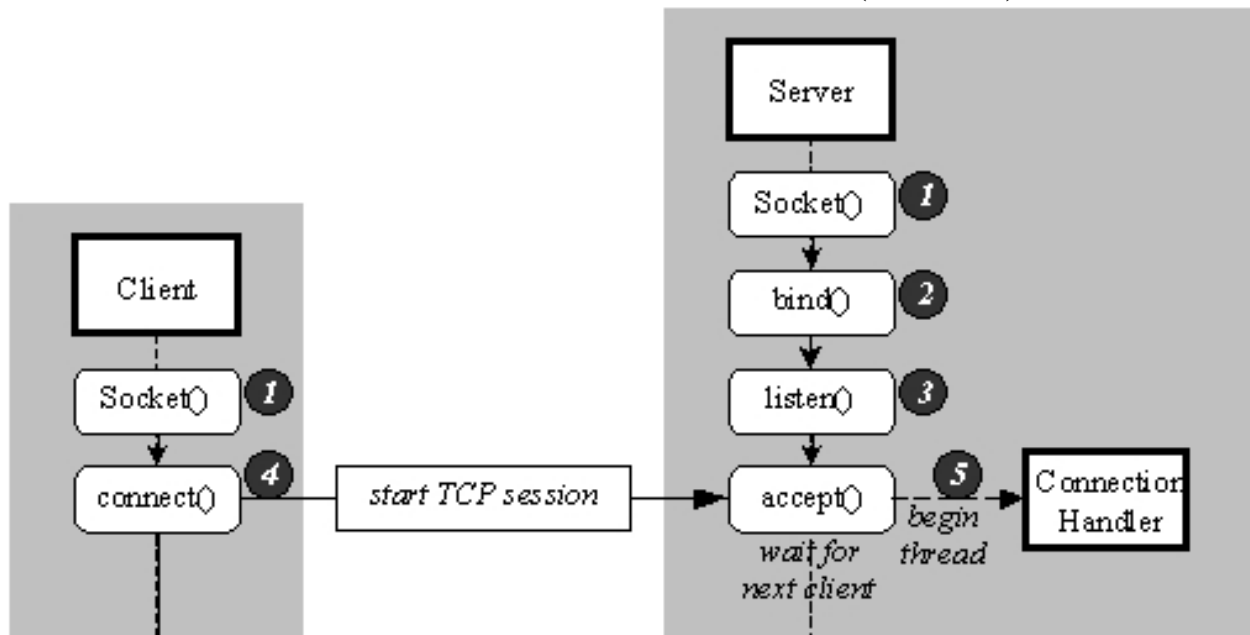
Interacțiune client-server - interacțiunea unui *socket* de tip flux pentru o tranzacție simplă.



Scenariu client-server bazat pe socket-uri

Clientul trimite un *mesaj* și așteaptă *raspunsul* la el. **Serverul** primește *mesajul* și generează *raspunsul*. Să urmărim pașii care duc la realizarea acestei tranzacții:

1. **Crearea punctelor terminale ale socket-ului.** Fiecare proces care folosește *socket-uri* trebuie mai întâi să creeze și să initializeze un *socket* folosind apelul **socket()**.
2. **Stabilirea unui port de servire.** Procesele serverului trebuie să lege (**bind**) *socket-urile* lor cu un nume de port unic ca să devină cunoscute în rețea.
3. **Așteptarea sosirii cererilor de conexiune.** Procesele serverului folosind *socket-urile* de tip flux trebuie să inițieze un apel **listen()** pentru a indica că sunt *gata de a accepta conexiuni de la clienți*. Serverul este acum deschis pentru comunicare prin acest *socket*. Apelul definește în mod tipic dimensiunea cozii pentru cererile care vin. Cererile suplimentare sunt ignorate de server.

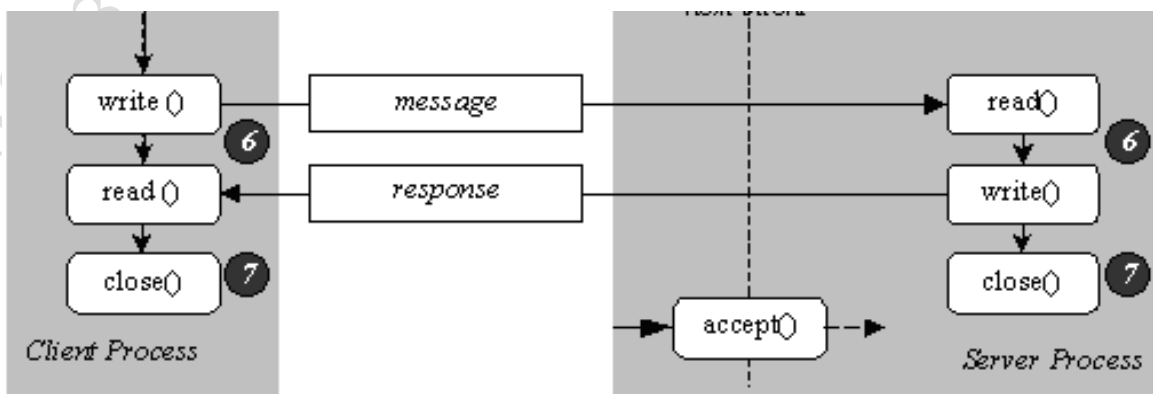


4. Conectarea la server. Clientul initiaza un apel **connect()** printr-un *socket* flux pentru a initia o conexiune cu portul care are serviciu. Clientul poate fi blocat optional, pana cand conexiunea este acceptata de server. La un raspuns favorabil, *socket-ul* client este asociat cu conexiunea la server.

5. Acceptarea conexiunii. Partea serverului accepta conexiunea cu un *socket* flux printr-un apel **accept()**. Apelul se va bloca, optional, daca nici o conexiune nu este ceruta. Daca sunt cerute multe conexiuni va fi stabilita prima din coada. Intr-un mediu cu mai multe fire de executie (multifilar), serverul porneste un nou fir de executie pentru a trata cererea clientului, pe un *socket* separat. Firul original redirecteaza apelul **accept()** catre firul de executie creat pentru tratarea conexiunii, redevenind disponibil pentru preluarea noilor cereri de servire de la noi clienti, pe acelasi *socket*.

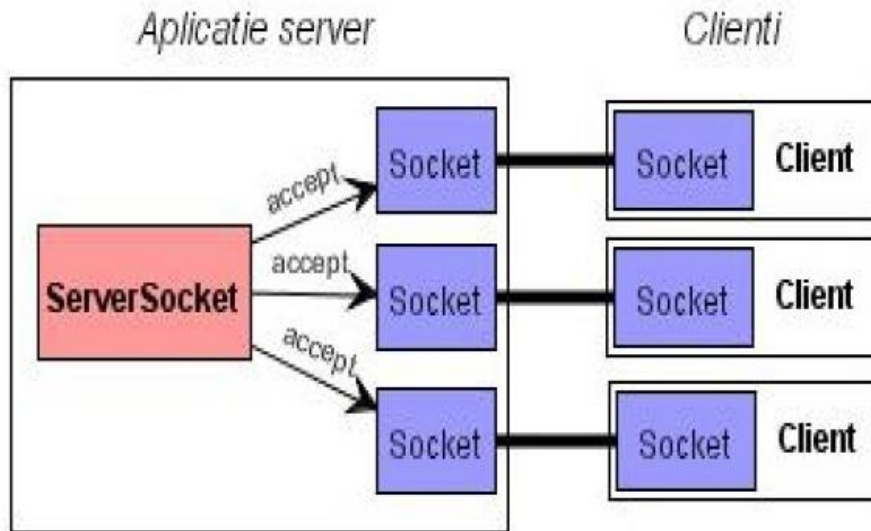
6. Utilizarea conexiunii. Clientii si serverele fac multe apeluri printre care si schimburile lor de informatii. **Interactiunile flux folosesc**, de obicei, apeluri **read()** and **write()**. **Socket-urile datagrama folosesc** apeluri **send()** and **receive()**. In scenariul nostru clientul initiaza un **write()** pentru a scrie mesajul. Serverul initiaza un **read()** pentru a primi mesajul. Apoi, initiaza un **write()** pentru a intoarce raspunsul la mesaj. Clientul initiaza un **read()** pentru a obtine raspunsul.

7. Inchiderea socket-ului. *Socket-urile* sunt resurse limitate. Ele trebuie sa fie reutilizate. Clientul si serverul initiaza, ambii, comenzi **close()** pentru a termina partea lor din *socket*. Firul de executie original al *socket-ului* server este insa inca activ si asteapta cereri de la clienti.



Observatii:

- **Socket-urile datagramme** necesita comenzi mai putine. Serverele datagramme *nu initiaza* apeluri **listen()** sau **accept()** pentru ca nu trebuie sa gestioneze sesiuni.
- Datagrammele folosesc primitivele de transfer, denumite **send()** si **receive()** in loc de **read()** si **write()**. (**read** si **write** sunt asociate cu fluxurile, deci si cu socket-urile flux)



2.2.1.1. Clase si metode pentru programarea cu sockets (TCP)

Pentru a putea realiza un program care utilizeaza socket-uri trebuie sa importam pachetele `java.io` si `java.net`

- **java.net.Socket** - Socket client TCP
 - o `public Socket(String host, int port) throws UnknownHostException, IOException` - constructor care deschide o conexiune TCP catre host-ul si portul specificati ca parametri.
 - o `public Socket(InetAddress host, int port) throws UnknownHostException, IOException` - constructor care deschide o conexiune TCP catre host-ul si portul specificati ca parametri.
 - o `public OutputStream getOutputStream()` - intoarce fluxul de iesire pentru socket.
 - o `public InputStream getInputStream()` - intoarce fluxul de intrare pentru socket.
 - o `public void close()` - inchide socket-ul.
- **java.net.ServerSocket** - Socket server TCP
 - o `public ServerSocket(int port) throws IOException, BindException` - constructor care inregistreaza serverul la portul specificat ca parametru.
 - o `public Socket accept()` - asculta cererile de conexiuni, intoarce un obiect Socket care va fi utilizat pentru comunicarea cu clientul.
- **java.io.DataOutputStream** - flux de intrare pentru un socket
 - o `public DataOutputStream(OutputStream out)` - constructor
 - o `public final void writeBytes(String s)` - scrie un sir catre socket
- **java.io.InputStreamReader** - citeste octeti de la un socket si îi transformă în caractere. care sunt apoi trimise catre un `BufferedReader`

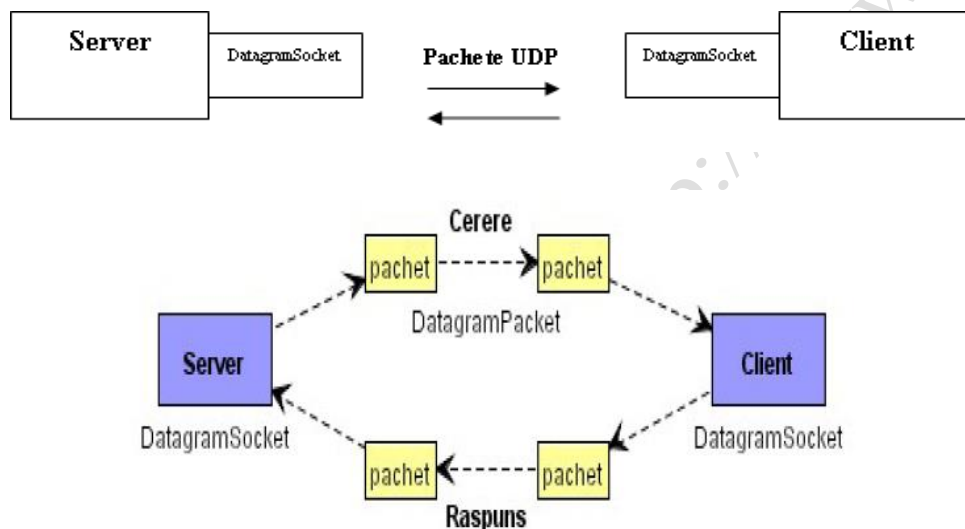
Retele de calculatoare – Informatica anul 3 (2019-2020)

- **java.io.BufferedReader** – cititor de secvențe de caractere cu zonă tampon
 - `public String readLine()` – citește următoarea linie de text dintr-un flux

2.2.2. UDP (User Datagram Protocol)

UDP este un protocol neorientat spre conexiune care transmite datele cu ajutorul protocolului IP. UDP oferă aplicațiilor acces direct la serviciul de transmitere a datelor dar nu oferă mecanisme de corectare a erorilor. Spre deosebire de TCP, UDP nu realizează o conexiune logică între cele două gazde, ci încapsulează informația în pachete independente (datagrame), împreună cu adresa destinație și numărul portului, și apoi le transmite prin rețea.

Atât clientul cât și serverul folosesc obiecte de tipul `DatagramSocket`.



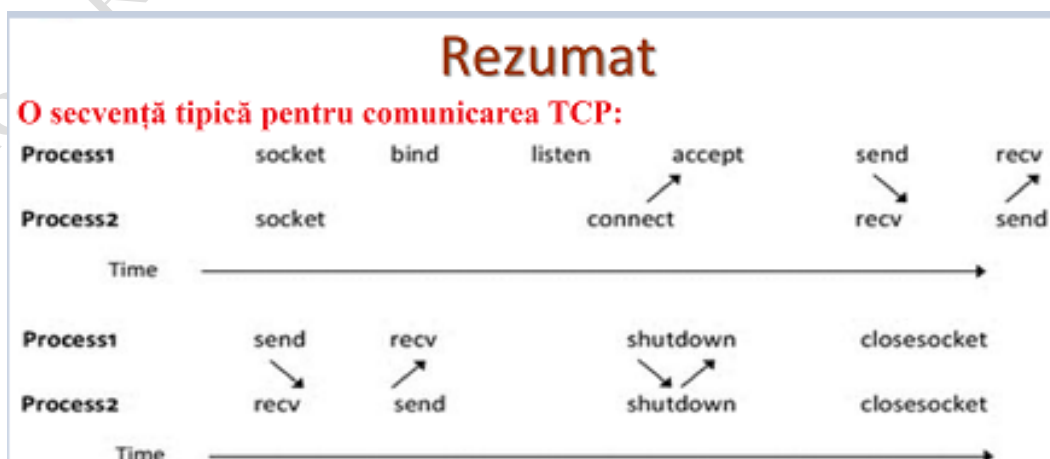
2.2.2.1. Clase și metode pentru programarea cu sockets (UDP)

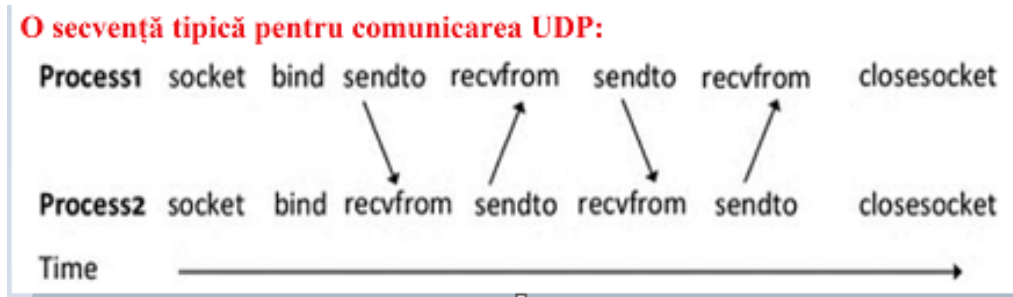
Pentru a putea realiza un program care utilizează socket-uri trebuie să importăm pachetele `java.io` și `java.net`

- **java.net.DatagramPacket** - clasa pentru realizarea pachetelor pentru transmiterea datelor
 - `DatagramPacket(byte inbuf[], int buflen)` - construiește un `DatagramPacket` pentru recepționarea datagramelor. Parametrul `inbuf` este un tablou de bytes pentru salvarea datelor primite și `buflen` indică numărul de octeți care vor fi citiți.
 - `DatagramPacket(byte inbuf[], int buflen, InetAddress iaddr, int port)` - construiește un pachet pentru transmiterea datelor. Fata de datagramele pentru recepție acest constructor specifică și adresa IP a mașinii destinație și numărul portului.
- **java.net.DatagramSocket** - clasa pentru contruirea socketurilor pentru recepția și transmiterea datagramelor.
 - `DatagramSocket()` throws `SocketException` - constructor care creează un socket utilizând primul port disponibil
 - `DatagramSocket(int port)` throws `SocketException` - constructor care creează un socket utilizând portul specificat ca parametru.
 - `void send(DatagramPacket p)` throws `IOException` - trimite o datagramă

Rețele de calculatoare – Informatica anul 3 (2019-2020)

- o `synchronized void receive(DatagramPacket p) throws IOException` – primește o datagrama
- o `synchronized void close()` – închide socketul
- o `int getLocalPort()` – întoarce portul pe care asculta socketul pentru datagrama
- **`java.io.DataInputStream`** - flux pentru citirea datelor de tip primitiv într-un format independent de masina pe care se lucreaza
 - o `public DataInputStream(InputStream in)` – constructor
 - o `readBoolean()`
 - o `readByte()`
 - o `readChar()`
 - o `readDouble()`
 - o `readFloat()`
 - o `readInt()`
 - o `readLong()`
 - o `readShort()`
 - o `readUnsignedByte()`
 - o `readUnsignedShort()`
 - o `String readUTF()`
- **`java.io.DataOutputStream`** - flux pentru scrierea datelor de tip primitiv într-un format independent de masina pe care se lucreaza
 - o `public DataOutputStream(OutputStream out)` – constructor
 - o `writeBoolean(boolean v)`
 - o `writeByte(int v)`
 - o `writeChar(int v)`
 - o `writeDouble(double v)`
 - o `writeFloat(float v)`
 - o `writeInt(int v)`
 - o `writeLong(long v)`
 - o `writeShort(int v)`
 - o `writeBytes(String s)`
 - o `writeChars(String s)`
 - o `writeUTF(String str)`
- **`java.io.ByteArrayInputStream`** – flux pentru citirea informatiilor care este creat pe un array de bytes existent.
- **`java.io.ByteArrayOutputStream`** – flux pentru scrierea informatiilor care este creat pe un array de bytes existent.
 - o `toByteArray()` – creaza un array de bytes. Dimensiunea array-ului este data de dimensiunea fluxului de iesire si el va contine octetii din buffer.





2.2.3. Trimiterea de mesaje catre mai multi clienti

Diverse situatii (exemplu: OSPF – DR(Designated Router)/BDR(Back-up Designated Router)/DROthers) impun gruparea mai multor clienti astfel încât un mesaj (pachet) trimis pe adresa grupului sa fie receptionat de fiecare dintre acestia. Gruparea mai multor programe în vederea trimiterii multiple de mesaje se realizeaza prin intermediul unui socket special, descris de clasa `MulticastSocket`, extensie a clasei `DatagramSocket`. Un grup de clienti abonati pentru trimitere multipla este specificat printr-o adresa IP din intervalul 224.0.0.1 – 239.255.255.255 si un port UDP. Adresa 224.0.0.0 este rezervata si nu trebuie folosita. http://en.wikipedia.org/wiki/Multicast_address

2.3. Fire de executie

2.3.1. Clasa `java.lang.Thread` - reprezinta un fir de executie

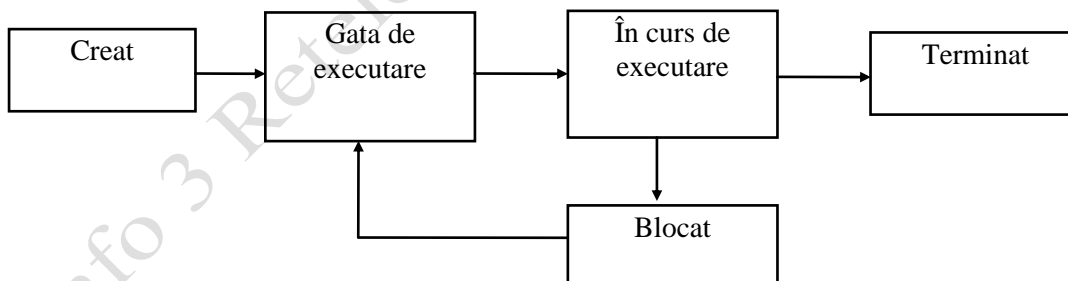
O primă modalitate de a crea și lansa în executare fire executie este de extinde clasa `Thread`

```
public class Thread extends Object implements Runnable
```

Pentru a crea un fir de executare, trebuie început prin a crea un obiect de tipul `Thread`:

```
Thread fir = new Thread();
```

Când firul este gata de executare, se invoca metoda sa `start()` fara argumente. Ca urmare firul de executie începe să execute metoda `run()`. Când metoda `run()` se încheie, firul de executie se termina. Un fir de executie trece prin urmatoarele stari:



Un fir este *gata de executare* daca îndeplineste toate conditiile pentru a se trece la executarea sa, dar nu i s-a alocat încă un procesor; când un procesor liber preia firul, acesta trece în starea *în curs de executare*. *Blocarea* unui fir de executare poate fi realizata pe baza unei conditii (de exemplu un semafor); ca urmare firul trece în starea *blocat*. La deblocarea sa, realizată prin mecanisme dintre care unele vor fi prezentate în continuare, el revine în starea de executare, așteptând ca un procesor să devină liber și să reia executarea sa.

Retele de calculatoare – Informatica anul 3 (2019-2020)

Metoda `run()` din clasa `Thread` nu implementează nici un comportament. De aceea trebuie ca programatorul să extindă clasa `Thread` și să rescrie metoda `run()`, precizând acțiunea dorită.

Interfața `java.lang.Runnable`

Utilizarea interfeței `Runnable` constituie o alternativă la extinderea clasei `Thread`. Avantajul constă în primul rând în însuși faptul că este o interfață: o clasă oarecare poate implementa `Runnable` și extinde o altă clasă (pe când o clasă ce extinde `Thread` nu mai poate extinde vreo altă clasă).

```
class Executie implements Runnable {
    . . .
    public void run() { . . . }
    . . .
}
class Clasa {
    . . .
    Executie ex = new Executie(...);
    Thread fir = new Thread(ex);
    fir.start();
    . . .
}
```

Pentru a opri temporar executia unui fir de poate utiliza metoda

```
static void sleep(long millis)
```

2.3.2. Sincronizarea firelor de executie

Pana acum am vazut cum putem crea fire de executie independente, care nu depind in nici un fel de executia sau de rezultatele altor fire. Exista insa numeroase situatii cand fire de executie separate, dar care ruleaza concurrent, trebuie sa comunice intre ele pentru a accesa diferite resurse comune (*sectiune critica*) sau pentru a-si transmite dinamic rezultatele "muncii" lor.

2.3.3. Primitive de sincronizare

Principalele primitive (metode) de sincronizare puse la dispoziție de Java sunt metode publice ale clasei `Object`:

- **`final void wait()`** – Fie `F` firul de executie din care este invocată una dintre aceste metode și fie `ob` obiectul (monitorul) curent. Executarea metodei suspendă firul `F`, adăugându-l listei de așteptare atașate monitorului `ob`; în același timp monitorul este "eliberat", fiind gata să ofere controlul asupra sa unuia dintre (eventualele) fire din lista sa de așteptare. Firul `F` rămâne în mulțimea de așteptare a monitorului până când este îndeplinită una dintre următoarele condiții:
 - un alt fir de executare invocă metoda `notify` prin intermediul obiectului `ob` și firul `F` este cel ales pentru a prelua controlul asupra monitorului
 - un alt fir de executare invocă metoda `notifyAll` prin intermediul obiectului `ob`

În continuare firul `F` așteaptă ca monitorul să devină liber (invocarea metodelor `notify` și `notifyAll` nu eliberează monitorul), după care el revine în starea dinaintea invocării lui `wait`

- **`final void notify()`** – Metoda `notify` "trezește" unul dintre eventualele fire din mulțimea de așteptare asociată monitorului care a invocat metoda. Acesta devine candidat la a prelua controlul asupra monitorului conform mecanismului descris mai sus.
- **`final void notifyAll()`** – Metoda `notifyAll` diferă de `notify` prin aceea că "trezește" toate eventualele fire din mulțimea de așteptare asociată monitorului care a invocat metoda.

Metodele pot lansa excepția `IllegalMonitorStateException` dacă firul curent nu deține controlul asupra monitorului reprezentat de obiectul curent și în plus trebuie să fie invocate din interiorul unei metode *sincronizate*. O metoda declarata cu modificatorul *synchronized* blocheaza accesul la resursa critica atunci cand este accesata de un fir de executie, astfel incat nici un alt fir de executie sa nu o acceseze in acelasi timp. Platforma Java asociaza un monitor fiecarui obiect al unui program aflat in executie. Acest monitor va indica daca resursa critica este accesata de vreun fir de executie sau este libera. In cazul in care este accesata, va pune un lacat pe aceasta, astfel incat sa impiedice accesul altor fire de executie la ea. In momentul cand resursa este eliberata, lacatul va fi eliminat, pentru a permite accesul altor fire de executie.

3. Partea practica (TEMA: pag.20)

3.1 Aplicatia A1 (Solutie Java): Inregistrarea unui client într-un grup

Indicatii:

```
import java.net.*;
import java.io.*;

public class MulticastClient {

    public static void main(String[] args) throws IOException {

        //adresa IP si portul care reprezinta grupul de clienti
        InetAddress group = InetAddress.getByName("230.0.0.1");
        int port=4444;

        MulticastSocket socket = null;
        byte buf[];

        try {
            //Se alatura grupului aflat la adresa si portul specificate
            socket = new MulticastSocket(port);
            socket.joinGroup(group);

            //asteapta un pachet venit pe adresa grupului
            buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);

            System.out.println(new String(packet.getData()));

        } finally {
            socket.leaveGroup(group);
            socket.close();
        }
    }
}
```

3.2. Aplicatia A2 (Solutie Java): Transmiterea unui mesaj catre un grup

Indicatii:

```
import java.net.*;
import java.io.*;

public class MulticastSend {

    public static void main(String[] args) throws Exception {

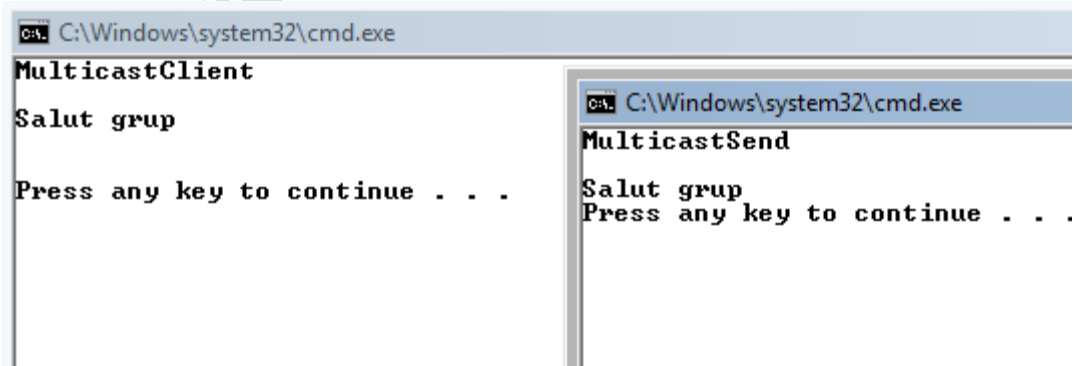
        InetAddress group = InetAddress.getByName("230.0.0.1");
        int port = 4444;
        byte[] buf;
        DatagramPacket packet = null;

        //Creeaza un socket cu un numar oarecare
        DatagramSocket socket = new DatagramSocket(0);

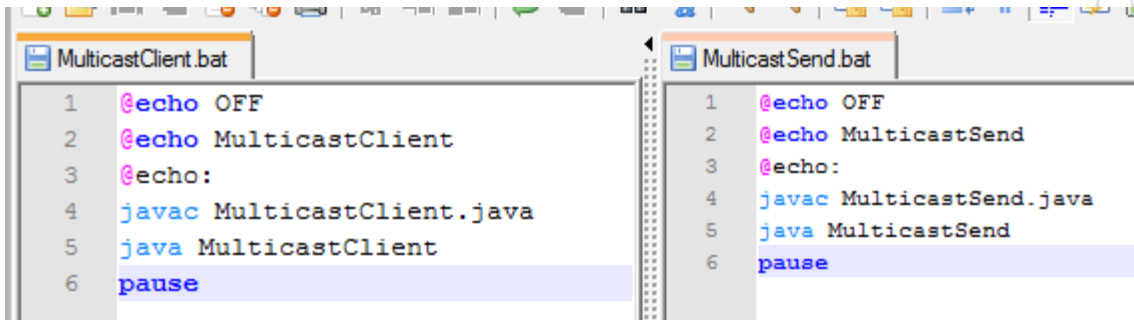
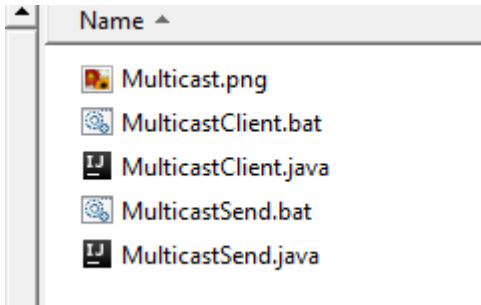
        try
        {
            //Trimite un pachet catre toti clientii din grup
            buf = (new String("Salut grup")).getBytes();
            packet = new DatagramPacket(buf, buf.length, group, port);
            socket.send(packet);

        } finally {
            socket.close();
        }
    }
}
```

Fisierul Multicast.png:



Folderul **A1-A2_Nume_Prenume:**



3.3. Aplicatia A3 (Solutie Java): Server multi-thread in Java

Server multi-thread de mai jos intoarce mesajele primite de la clienti impreuna cu un antet. Aplicatia este implementata in trei clase: ServerApp, ServerThread si Client. ServerThread este o clasa interna care extinde clasa Thread. Pentru fiecare cerere de conexiune venita din partea unui client, este lansat un nou fir de executie.

Indicatii:

```
/*
 * Clasa ServerApp
 */

import java.io.*;
import java.net.*;

public class ServerApp {

    private ServerSocket serverSocket;
    private int port;
```

```

public ServerApp(int port) {
    this.port = port;
    try {
        serverSocket = new ServerSocket(port);
    } catch(IOException ioe) {
        System.out.println(ioe.getMessage());
    }
    while(true) {
        try {
            Socket socket = serverSocket.accept();
            new ServerThread(socket).start();
        } catch(IOException ioe) {
            System.out.println(ioe.getMessage());
        }
    }
}

public static void main(String[] args) {
    ServerApp server = new ServerApp(15876);
}

}

/*
 * Clasa interna ServerThread
 */

class ServerThread extends Thread {

    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    private String nume;

    public ServerThread(Socket socket) {
        this.socket = socket;
        try {
            in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true);
        } catch(IOException ioe) {
            System.out.println(ioe.getMessage());
        }
    }
}

```



```

public void run() {
    int contor = 0;

    try{
        nume = in.readLine();
    }catch(IOException ioe){
        System.out.println(ioe.getMessage());
    }

    while(ecouMesaj(contor)) {
        contor++;
    }
}

private boolean ecouMesaj(int contor) {
    try {
        String mesaj = in.readLine();
        if(!mesaj.equals("EXIT")){
            System.out.println("Mesaj "+contor+" (" +nume+" ):"+mesaj);
            out.println("Mesaj " + contor + ": " + mesaj);
            return true;
        }
        else{
            System.out.println("Inchidere conexiune");
            socket.close();
            return false;
        }
    }catch(IOException ioe) {
        System.out.println(ioe.getMessage());
        System.exit(0);
        return false;
    }
}
}

```



```

/*
 * Clasa Client
 */
import java.io.*;
import java.net.*;

public class Client {

    private InetAddress host;
    private int port;
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    private BufferedReader input;
    private String mesaj;
    private String raspuns;
    private String nume;

    public Client(int port,String nume) {
        this.port = port;
        this.nume = nume;
        try {
            host = InetAddress.getLocalHost();
            socket = new Socket(host, port);
            in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true);
            input=new BufferedReader(new InputStreamReader(System.in));
        } catch(UnknownHostException uhe) {
            System.out.println(uhe.getMessage());
        } catch(IOException e) {
            System.out.println(e.getMessage());
        }
        out.print(nume+"\n");

        while(true){
            solicitaServer();
        }
    }
}

```

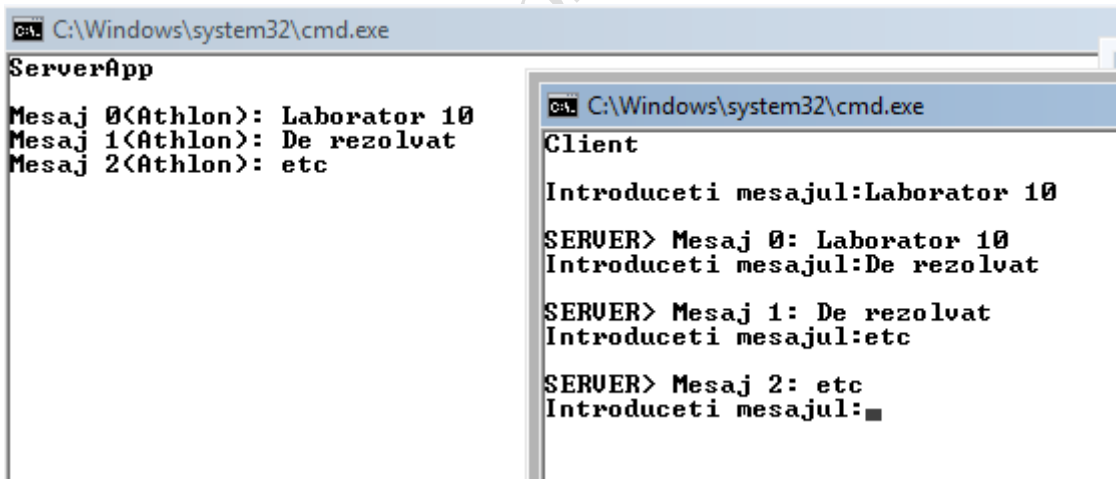
```
private void solicitaServer() {
    try {
        System.out.print("Introduceti mesajul:");
        mesaj = input.readLine();
        out.println(mesaj);

        if(mesaj.equals("EXIT")) {
            System.out.println("Inchidere conexiune");
            socket.close();
            System.exit(1);
        }

        raspuns = in.readLine();
        System.out.println("\nSERVER> " + raspuns);
    } catch(IOException e) {
        System.out.println(e.getMessage());
    }
}

public static void main(String[] args) {
    new Client(15876,"Athlon");
}
}
```

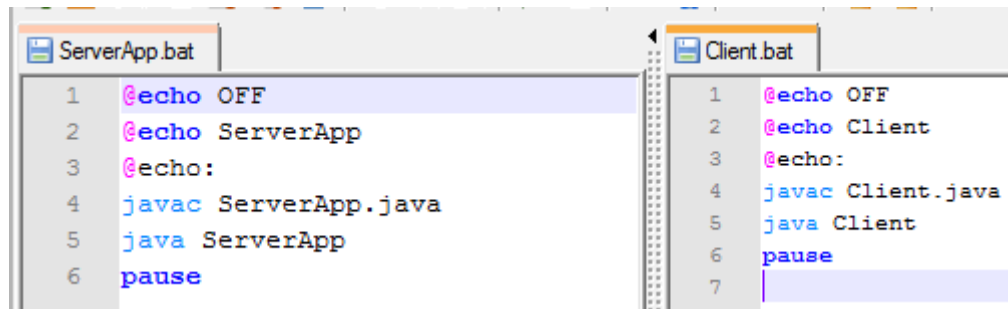
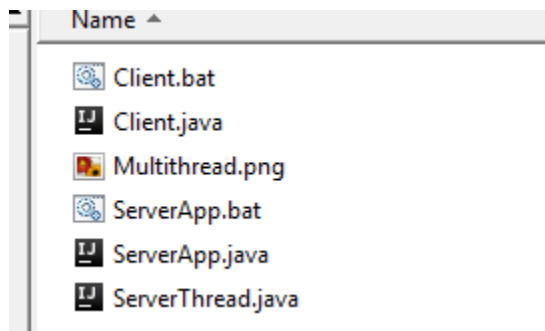
Fisierul Multithread.png:



```
C:\Windows\system32\cmd.exe
ServerApp
Mesaj 0(Athlon): Laborator 10
Mesaj 1(Athlon): De rezolvat
Mesaj 2(Athlon): etc

C:\Windows\system32\cmd.exe
Client
Introduceti mesajul:Laborator 10
SERVER> Mesaj 0: Laborator 10
Introduceti mesajul:De rezolvat
SERVER> Mesaj 1: De rezolvat
Introduceti mesajul:etc
SERVER> Mesaj 2: etc
Introduceti mesajul:█
```

Folderul **A3_Nume_Prenume:**



3.4. Aplicatia A4 – Ghicire numar (Solutie multithread Java) Indicatii: [Multithread in Java](#)

Implementati jocul de ghicire (Laboratorul 9) a unui numar folosind un server-multithread. Atunci cand serverul este pornit el va salva un numar aleator intre 0 si 500.

Indicatii:

```
int randomNumber=(int) (Math.random()*500);
```

Jocul va incepe numai dupa ce cei trei clienti s-au conectat la server. Fiecare client va citi numere de la tastatura si va trimite aceste numere la server. Serverul va raspunde prin mesaje (MARE, MIC, CORECT). Jocul continua pana cand unul dintre clienti este declarat castigator.

3.5. Aplicatii de retea in Python (Solutii Python pentru 3.1, 3.2, 3.3, 3.4)

3.5.1 Aplicatia A5.1: Inregistrarea unui client într-un grup

3.5.2 Aplicatia A5.2: Transmiterea unui mesaj catre un grup

3.5.3 Aplicatia A5.3: Server multi-thread

3.5.4 Aplicatia A5.4: Ghicire numar (Solutie multithread) Indicatii: [Multithread in Python](#)

Challenge: Interfata grafica

Recomandare: Qt Designer , cu Designer din Anaconda prompt).

<http://pythonforengineers.com/your-first-gui-app-with-python-and-pyqt/>,

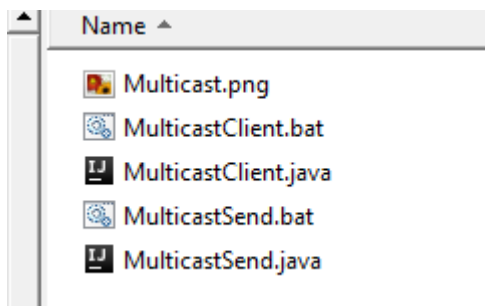
Indicatii: Recapitulare Python

- Python_intro (Lab_02, Lab_03)
- Programare_Python (Lab_02, Lab_03)
- Byte-of-python (Lab_02, Lab_03)
- Python socket network programming_1 (Lab_08)
- Python socket network programming_2 (Lab_08)
- Python Files and os.path (Lab_09)
- Programarea socket-urilor de retea in Python (Lab_09: BasicsOfSockets.pdf)
- [Multithread in Python](#) ([Multithread in Java](#))

Obs: Anexa 2 - The Programming Process (pag.24)

Observatie: (cod sursa, capturi prezentand functionarea acestora, comentarii dupa modelul celor de la pag.24, Anexa 1). Cursantii sunt incurajati sa dezvolte aplicatii proprii in contextul celor prezentate, adaugand comentarii si prezentand o analiza aprofundata privind solutiile posibile care se pot aborda.

Folderul **Aplicatie_Nume_Prenume** corespunzator fiecarei aplicatii va contine fisiere conform exemplului urmator:



4. Tema:

- Toate punctele din sectiunea 3 “partea practica” se vor relua de catre cursanti, folosind etapele de lucru indicate. Rezultatele experimentale:
 - L10_num+prenume_java (folder) : contine subfolderele 3.1., 3.2., 3.3., 3.4., fiecare subfolder cu fisierele pentru fiecare aplicatie (.java, [.bat](#), . png, insotite de un *readme.txt* pentru particularitati de rulare, conform prezentarilor facute). **Atentie la modul de prezentare din Anexa 1...asa ar trebui!...si de pe masini diferite: ip sursa != ip destinatie)**
 - **L10_num+prenume_Python (folder)** – cu subfolderele 3.3.1, 3.5.2, 3.5.3, 3.5.4 (fiecare din acestea contine scripturile .py si .doc/ .png (snipping tool) pentru aplicatiile Python.

Atentie la modul de prezentare din Anexa 1...asa ar trebui!..si de pe masini diferite: ip sursa != ip destinatie. RECOMANDARE: 3.6/7.1 (Lab2, Lab3, Lab4, Lab5, Lab6, Lab7, Lab8, Lab9)

se vor arhiva cu numele **L10_num+prenume_info3.rar** si se va trimite prin e-mail la adresa retelecdsd@gmail.com precizandu-se la subject: **L10_num+prenume_info3**, pana pe data de **13 decembrie 2019 e.n., ora 8.00 a.m.** (**Atentie, gmail nu "prea vrea" .rar in .rar** <http://www.makeuseof.com/tag/4-ways-email-attachments-file-extension-blocked>).

VARIANTE pentru trimiterea arhivei: <http://www.gfile.ro>; <http://www.wetransfer.com>

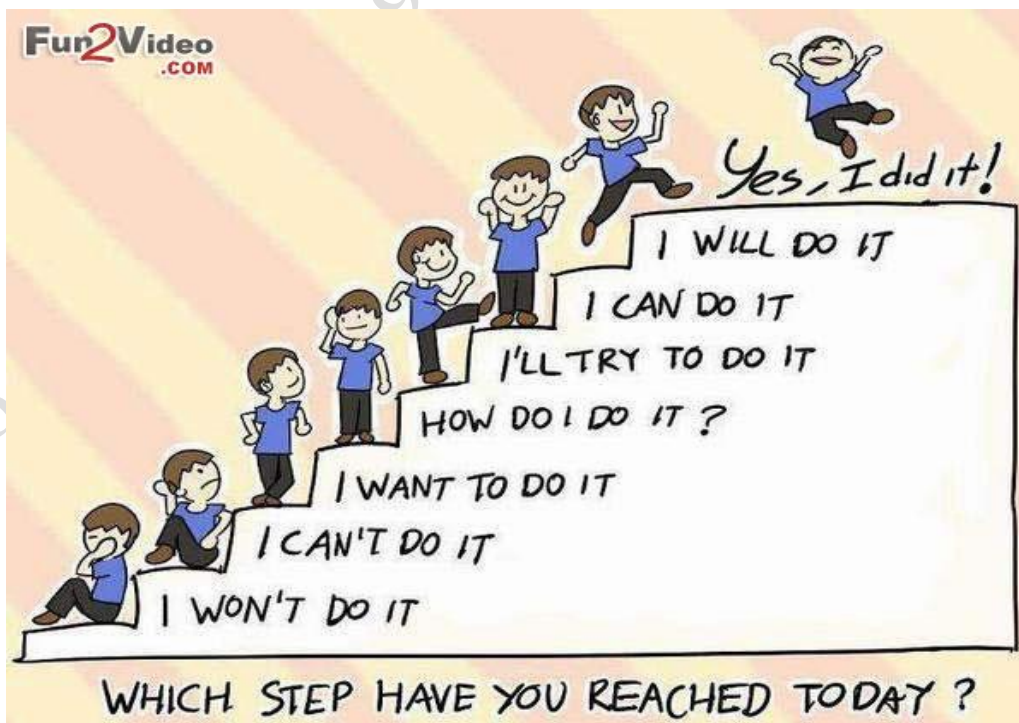
Cursantii sunt incurajati sa analizeze si sa comenteze rezultatele obtinute, studiind si materialele indicate in bibliografie si anexe. (+ **Recapitulare Laboratoarele 1+2+3+4+5+6+7+8+L9**) (**Pentru Modeler, varianta "programare" C++:** [OMNeT++ Network Simulation Framework](http://www.omnetpp.org/) <http://www.omnetpp.org/>;

Obs:

Punctaj maxim (Data trimiterii temei)			
<= 13.12. 2019	17.12. 2019	21.12.2019	25.12.2019
100 pct	80 pct	60 pct	50 pct

Obs: Participarea (activa!) la Curs si Laborator permite, prin cunostintele acumulate, obtinerea unor rezultate bune si f. bune, asa cum ni le dorim cu totii.

DE ANALIZAT [readme-ul readme_mod_work_dir.pdf](#) (si un numai!... de exemplu si [readme_lab_modeler.pdf](#)) de la adresa <http://www.cdsd.ro>



Sursa: <http://www.funnfun.in/wp-content/uploads/2013/06/steps-of-success-encouraging-quote.jpg>

How to send an e-mail

<http://lifehacker.com/5803366/how-to-send-an-email-with-an-attachment-for-beginners>
<https://support.google.com/mail/answer/6584?hl=en> “As a security measure to prevent potential viruses, Gmail doesn't allow you to send or receive executable files (such as files ending in .exe).”
<https://support.google.com/mail/answer/2480713?hl=en>
<http://fastupload.ro/free.php>
<http://www.computerica.ro/siteuri-transfer-fisiere-mari-upload/>

Bibliografie:

Lab_01, Lab_02, Lab_03, Lab_04, Lab_05, Lab_06, Lab_07, TL_01, TL_02, TL_03, TL_04
<http://www.cdsd.ro/cursuri>

<http://support.microsoft.com/kb/140859>
<http://www.windowsreference.com/windows-2000/how-to-add-static-route-in-windows-xp2000vista/>
http://www.comptechdoc.org/os/linux/usersguide/linux_ugrouting.html
<http://linux-ip.net/html/ch-routing.html>
http://www.3com.com/other/pdfs/infra/corpinfo/en_US/501302.pdf
<http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/route.mspx?mfr=true>

efg' Mathematics, <http://www.efg2.com/Lab/Mathematics/CRC.htm>

Java API, <https://docs.oracle.com/javase/8/docs/api/>

Java Tutorial, Writing Your Own Filtered Streams <http://www.rgagnon.com/javadetails/java-0416.html>

http://en.wikipedia.org/wiki/Cyclic_redundancy_check
<http://www34.brinkster.com/dizzyk/crc32.asp>
<http://www.createwindow.com/programming/crc32/crcfile.htm>
<http://webnet77.com/cgi-bin/helpers/crc.pl>
<http://www.softpedia.com/get/Others/Miscellaneous/CRC32-Calculator.shtml>
<http://www.wikiera.net/EthernetCRC-readytouseexample.html>
http://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html

Modeler Tutorials

https://rpmapps.riverbed.com/ae/4dcgi/SIGNUP_NewUser
<https://supportkb.riverbed.com/support/index?page=content&id=S24443>
https://rpmapps.riverbed.com/ae/4dcgi/DOWNLOAD_HOME
https://rpmapps.riverbed.com/ae/4dcgi/REG_TransactionCode

- Install Riverbed Modeler 17.5 Windows 10, 8.1, 8 and 7 (<https://www.youtube.com/watch?v=TpenN2jYbHQ>)
- Install Riverbed Modeler (<https://www.youtube.com/watch?v=DQ3XhHYuFGA>)
- How to activate riverbed modeler 17.5 (<https://www.youtube.com/watch?v=h-ImeJMqiSA>)

Rețele de calculatoare – Informatica anul 3 (2019-2020)

- How to solve invalid activation of Opnet Modeler 17.5 (<https://www.youtube.com/watch?v=13ZBcXkW46s>)
- Riverbed Modeler 17.5 Tutorial - Switched Lan (<https://www.youtube.com/watch?v=XdebwQLrr0w>)
- 6-Virtual LAN (VLAN) configuration in OPNET Riverbed (<https://www.youtube.com/watch?v=Ajz7bVO5WJM>)
- Riverbed Modeler Configuracion VLAN (<https://www.youtube.com/watch?v=rP3jPMcyEFk>)
- [Ethernet \(lab_04\)](#)
- Riverbed Opnet 17.5 Tutorial - The Ethernet network (https://www.youtube.com/watch?v=fS_J6ApFJtc)
- 6-Virtual LAN (VLAN) configuration in OPNET Riverbed (<https://www.youtube.com/watch?v=Ajz7bVO5WJM>)
- Riverbed Modeler Tutorial 3 Configuracion VLAN (<https://www.youtube.com/watch?v=rP3jPMcyEFk>)

Python (Lab1, Lab2)

Using Python on Windows - <https://docs.python.org/3/using/windows.html>

The Hitchhiker's Guide to Python - <http://docs.python-guide.org/en/latest/intro/learning/>

A Byte of Python - <https://www.gitbook.com/book/swaroopch/byte-of-python/details>

GUI Programming in Python - <https://wiki.python.org/moin/GuiProgramming>

<https://winpython.github.io/> ; <https://www.python.org/>

<https://social.technet.microsoft.com/wiki/contents/articles/910.windows-7-enabling-telnet-client.aspx>

<http://www.telnet.org/htm/places.htm>

rainmaker.wunderground.com : weather via telnet!

<https://docs.python.org/3/library/socket.html>

18.1. [socket](#) — Low-level networking interface

Java Sockets

<http://download.oracle.com/javase/tutorial/networking/sockets/>

<http://www.oracle.com/technetwork/java/socket-140484.html>

Python Sockets

<http://docs.python.org/howto/sockets.html>

C++ Sockets

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

<http://cs.baylor.edu/~donahoo/practical/CSockets/winsock.html>

PHP Sockets

<http://www.php.net/manual/en/book.sockets.php>

Perl Socket

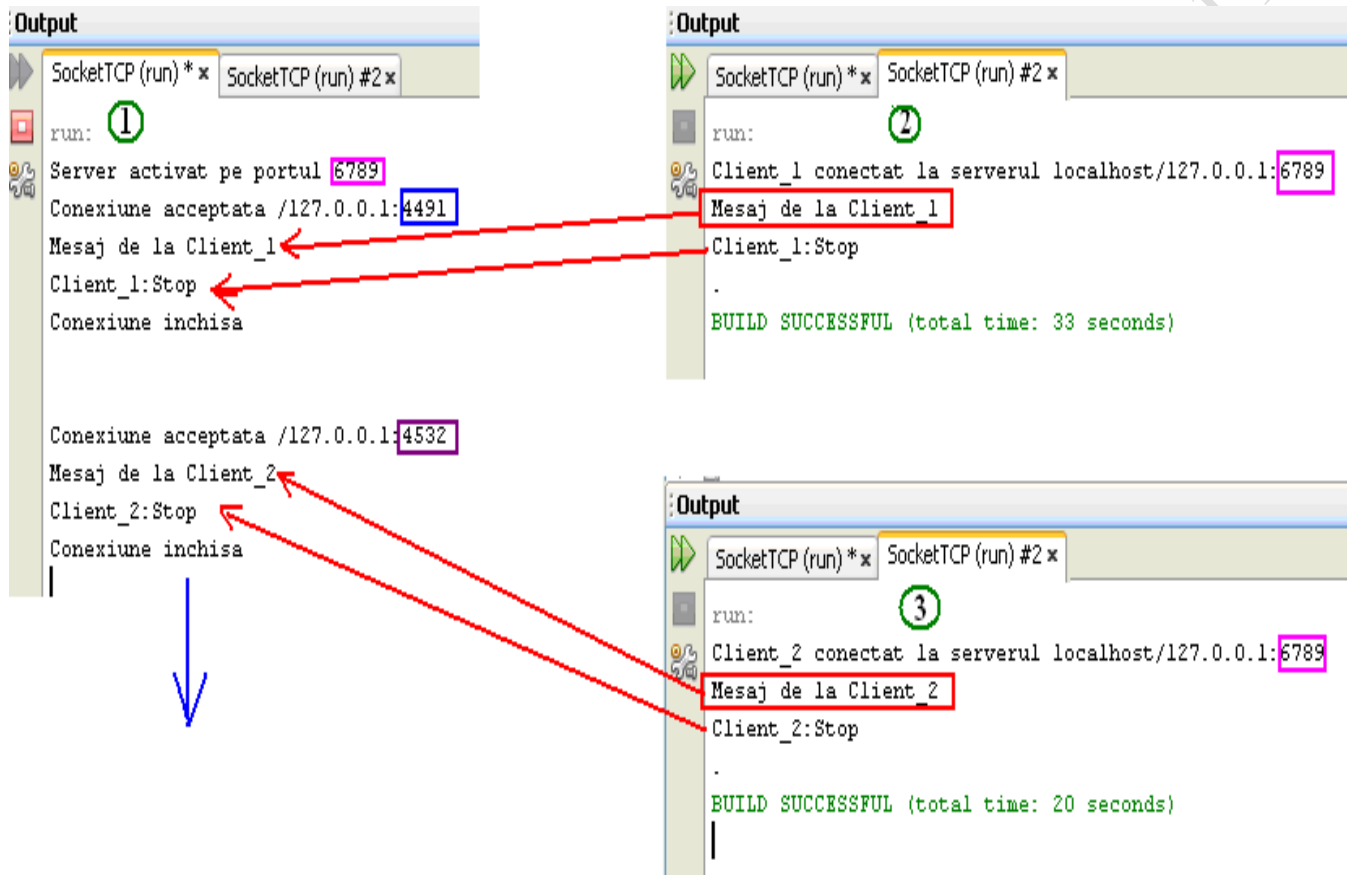
<http://www.devshed.com/c/a/Perl/Socket-Programming-in-PERL/>

Ruby Sockets

http://en.wikibooks.org/wiki/Ruby_Programming/Reference/Objects/Socket

etc....

Anexa 1 : Exemplu “comentariu”



Anexa 2: The Programming Process

1. Identify the Problem - **What** Are You Trying To Do?
 - Requirements
 - Specification
2. Design a Solution - **How** Is It Going To Be Done?
3. Write the Program - **Teaching** the Computer
 - Code
 - Compile
 - Debug
4. Check the Solution - **Testing** it Understands You