Sockets in Java

Daniel Hagimont

IRIT/ENSEEIHT
2 rue Charles Camichel - BP 7122
31071 TOULOUSE CEDEX 7

Daniel.Hagimont@enseeiht.fr
http://hagimont.perso.enseeiht.fr

What are sockets

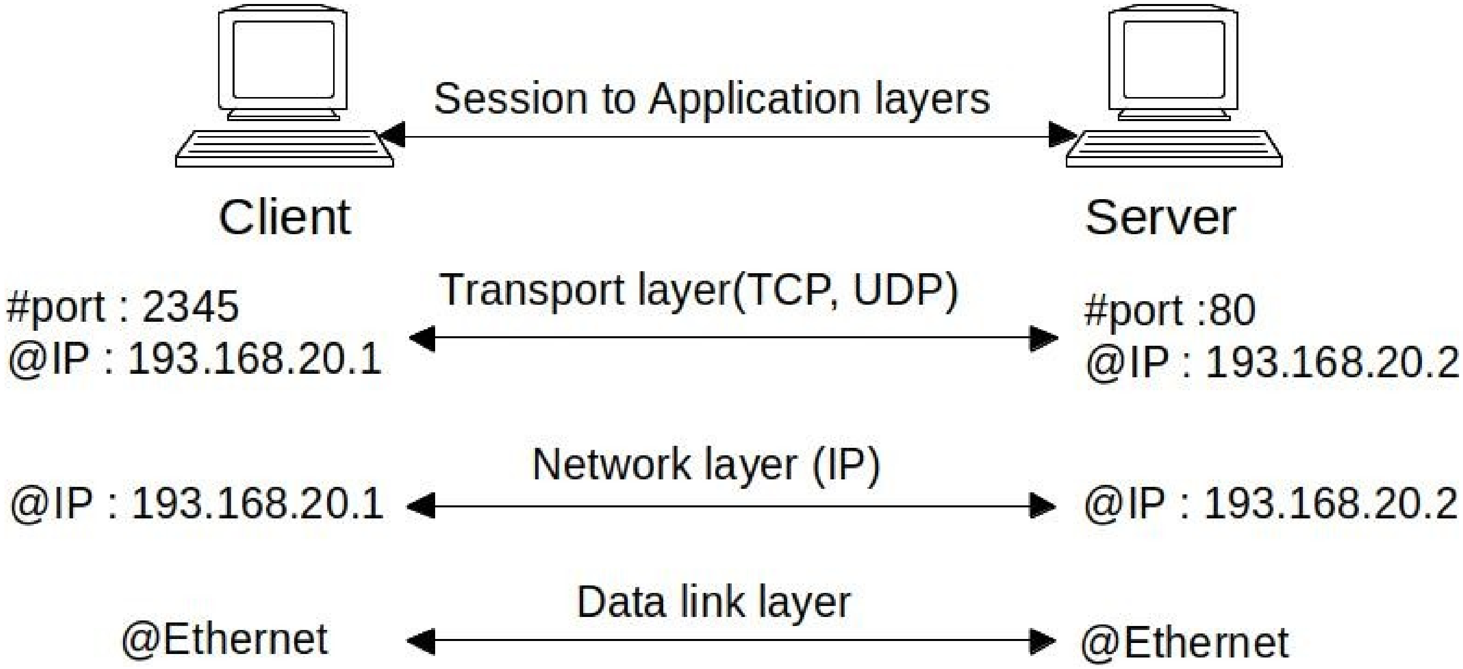
- Interface for programming network communication
- Allow building client/server applications
  - Applications where a client program can make invocations to server programs with messages (requests) rather than shared data (memory or files)
  - Example: a web browser and a web server
- Not only client/server applications
  - Example: a streaming applications (VOD)

Two modes connected/not connected

- Connected mode (TCP)
  - Communication problems are handled automatically
  - Simple primitives for emission and reception
  - Costly connection management procedure
  - Stream of bytes: no message limits
- Not connected mode (UDP)
  - Light weight: less resource consumption
  - More efficient
  - Allow broadcast/multicast
  - All communication problems (packet loss) have to be handled by the application

Sockets

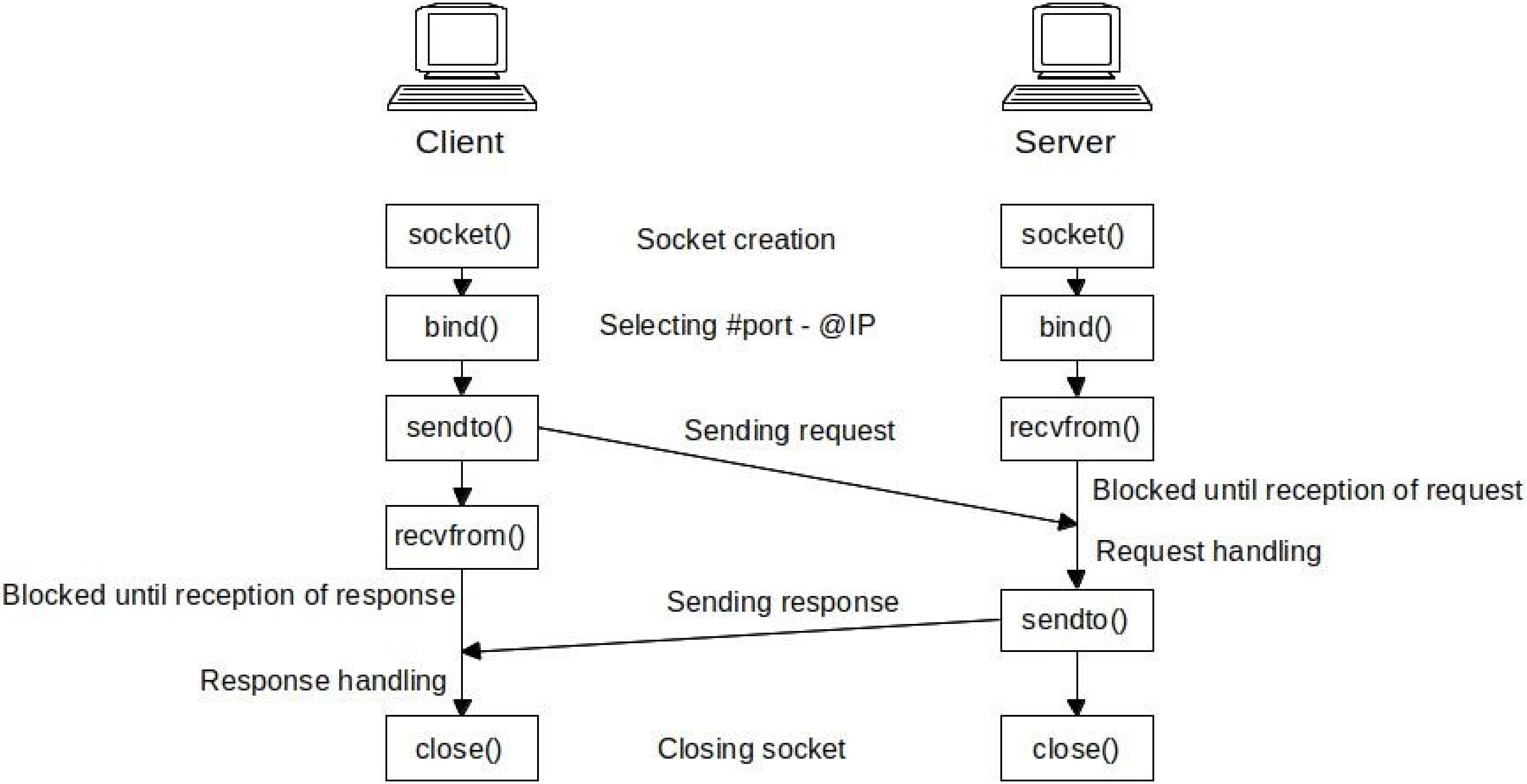
- Network access interface
- Developed in Unix BSD
- @IP, #port, protocol (TCP, UDP, ...)



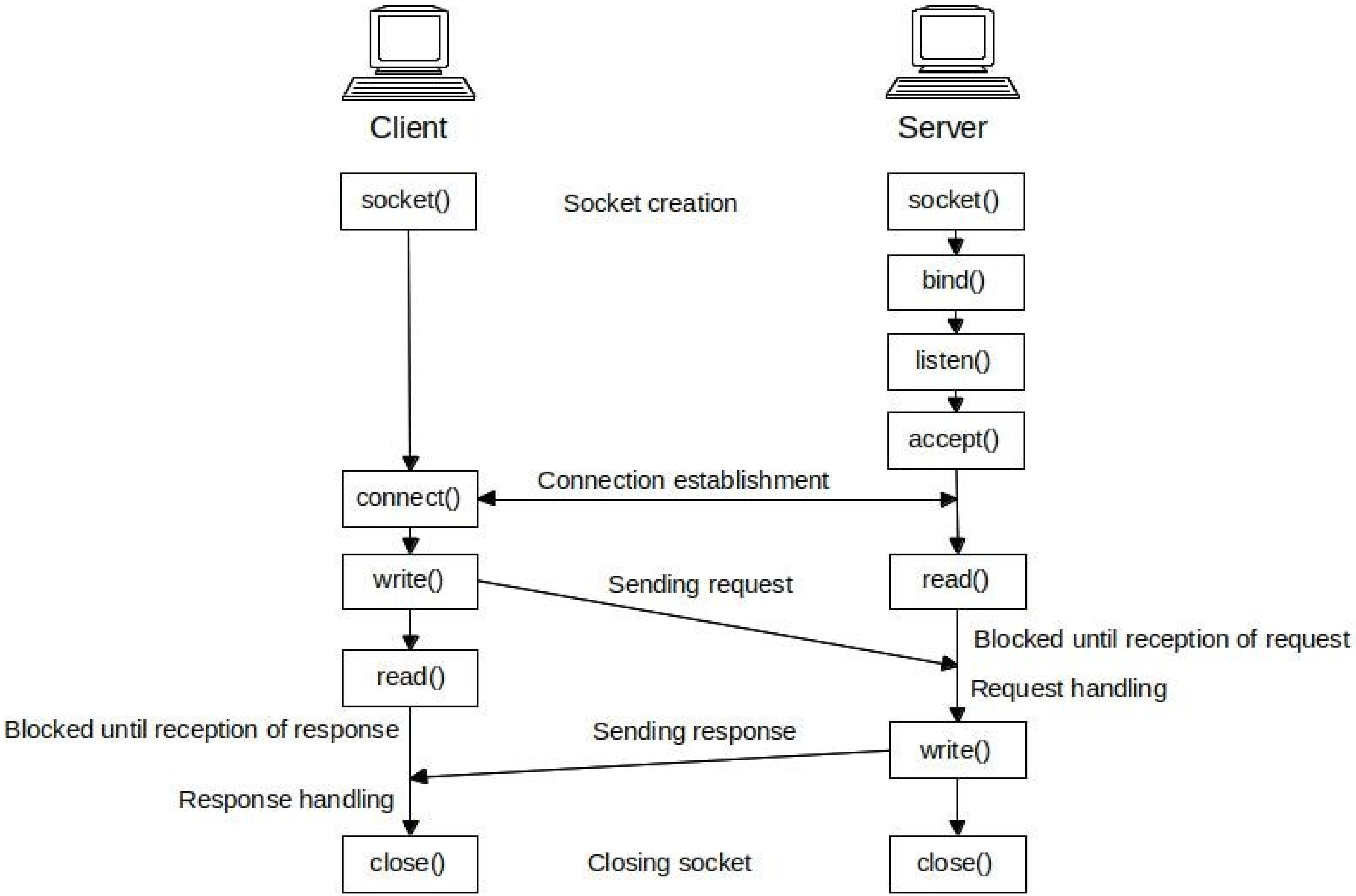
The socket API

- Socket creation: socket(family, type, protocol)
- Opening the dialog:
  - Client: bind(..), connect(...)
  - Server: bind(..), listen(...), accept(...)
- Data transfer:
  - Connected mode: read(...), write(...), send(...), recv(...)
  - Non-connected mode: sendto(...), recvfrom(...), sendmsg(...), recvmsg(...)
- Closing the dialog:
  - close(...), shutdown(...)

Client/Server in non-connected mode



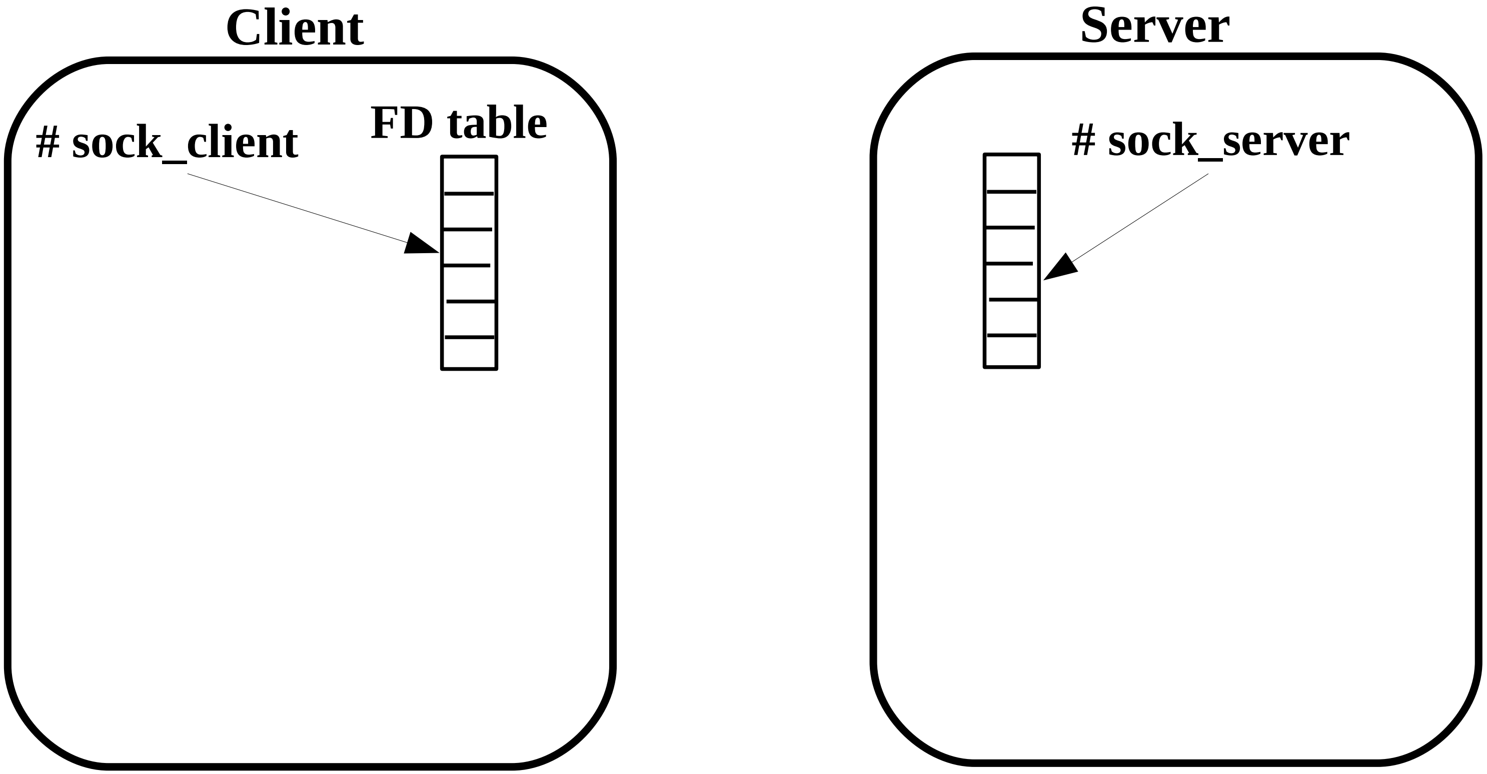
Client/Serveur in connected mode



socket() function

- int socket(int family, int type, int protocol)
- family
  - AF_INET: for Internet communications
  - AF_UNIX: for local communications
- type or mode
  - SOCK_STREAM: connected mode (TCP)
  - SOCK_DGRAM: non-connected mode (UDP)
  - SOCK_RAW: direct access to low layers (IP)
- protocol :
  - Protocol to use (different implementations can be installed)
  - 0 by default (standard)

After call to socket()

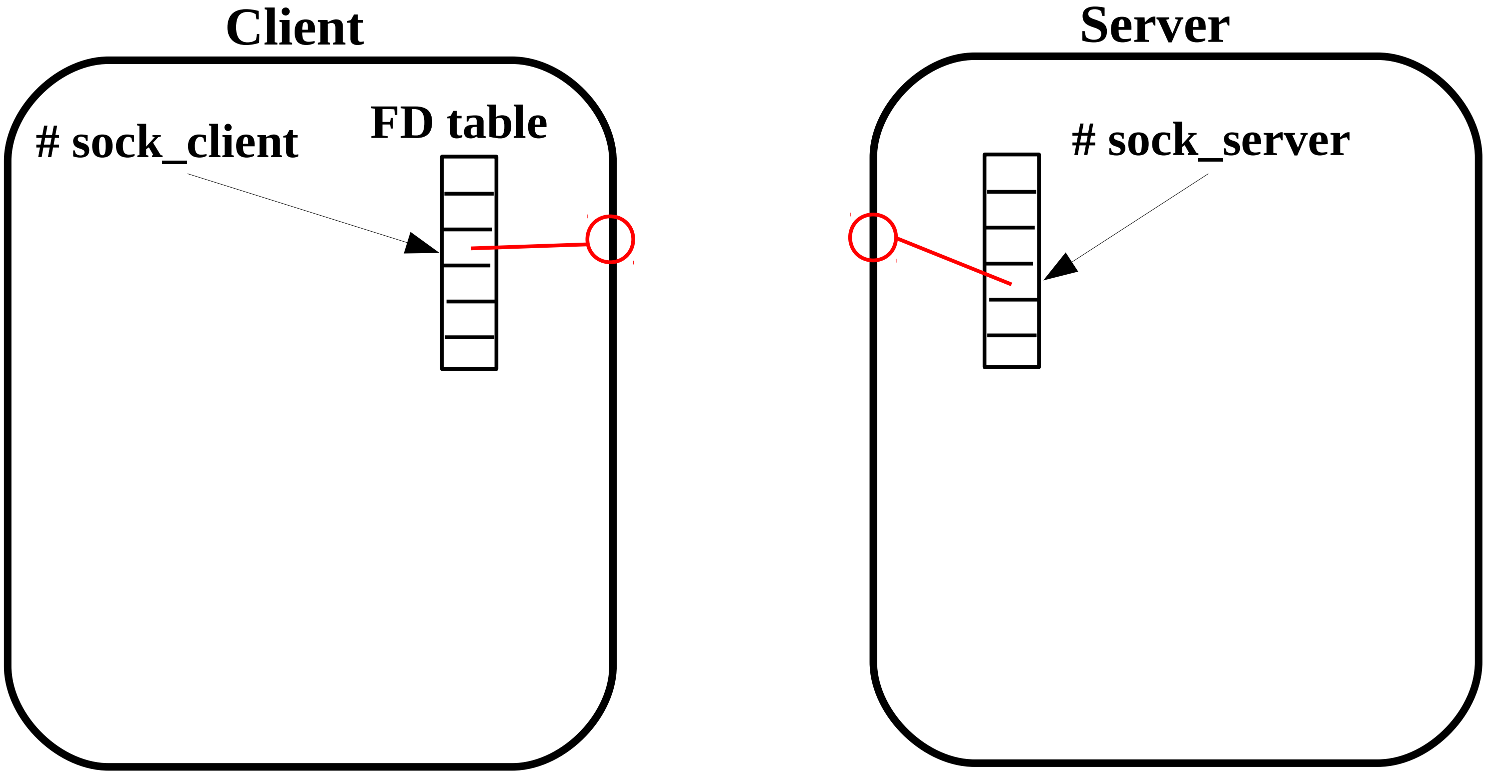


bind() function

- **int bind(int sock_desc, struct sockaddr *my_@, int lg_@)**
- **sock_desc: socket descriptor returned by socket()**
- **my_@: IP address and # port (local) that should be used**
- **Example (client or server):**

```
int sd;
struct sockaddr_in my_address; // @IP, #port, mode

sd = socket(AF_INET, SOCK_STREAM, 0);
my_address.sin_family = AF_INET;
my_address.sin_port = 0; // let system choose a port
my_address.sin_addr.s_addr = INADDR_ANY;
                         // any network interface

bind(sd, (struct sockaddr *)&my_address, sizeof(my_address));
```

After call to bind()



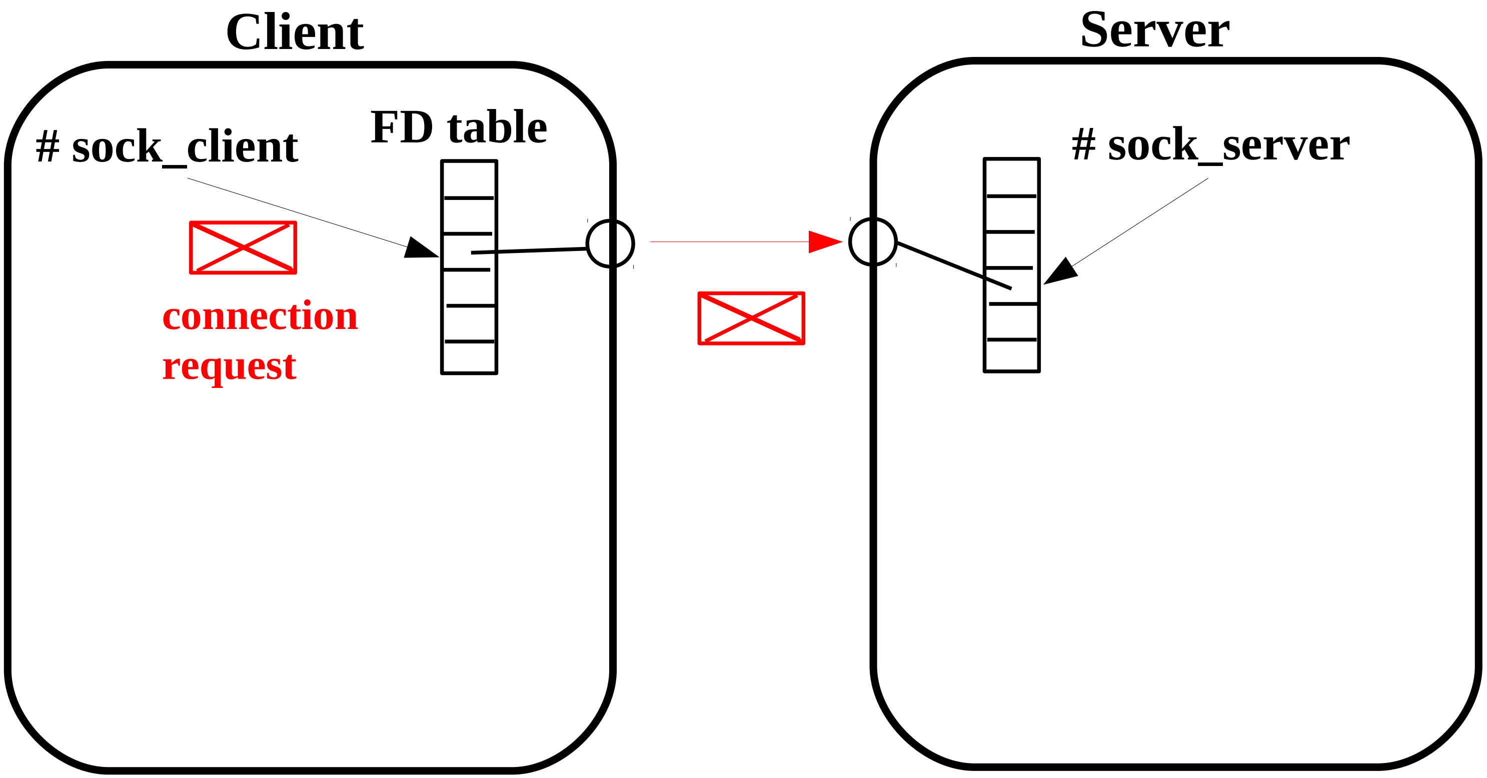
We can already exchange messages in non-connected mode

connect() function

- **int connect(int sock_desc, struct sockaddr * @_server, int lg_@)**
- **sock_desc: socket descriptor returned by socket()**
- **@_server: IP address and # port of the remote server**
- **Example of client:**

```
int sd;
struct sockaddr_in server; // @IP, #port, mode
struct hostent remote_host; // name et @IP

sd = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = htons(13);
remote_host = gethostbyname(“www.enseeiht.fr”); // DNS loookup
bcopy(remote_host->h_addr, (char *)&server.sin_addr,
       remote_host->hlength); // copy the address
connect(sd, (struct sockaddr *)&server, sizeof(server));
```

After call to connect()



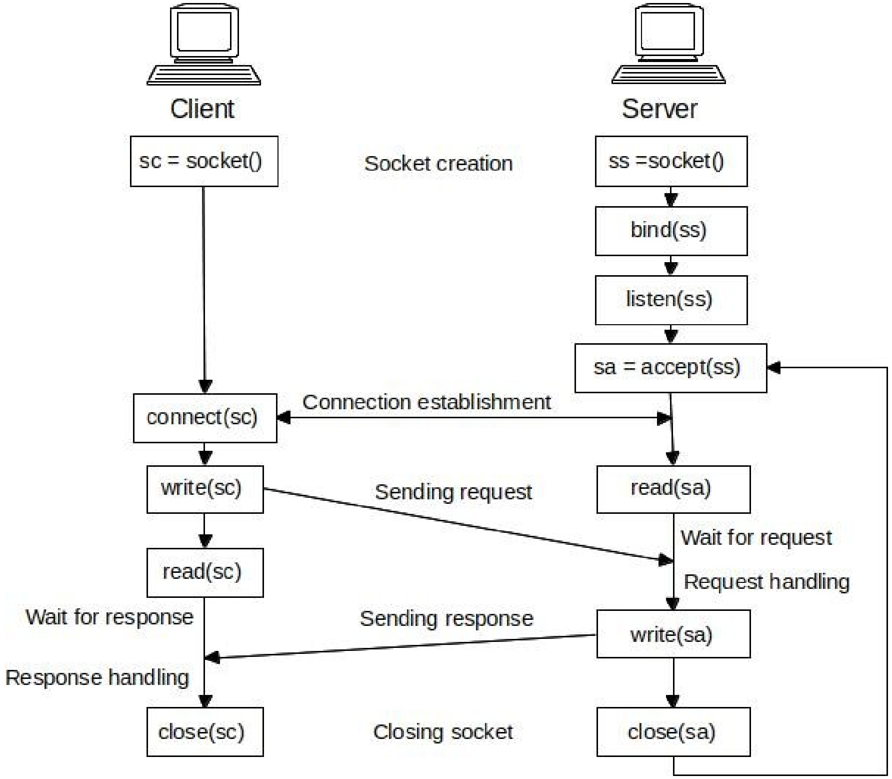
listen() function

- **int listen(int sock_desc, int nbr)**
- **sock_desc: socket descriptor returned by socket()**
- **nbr: maximum number of pending connections**
- **Example of server:**

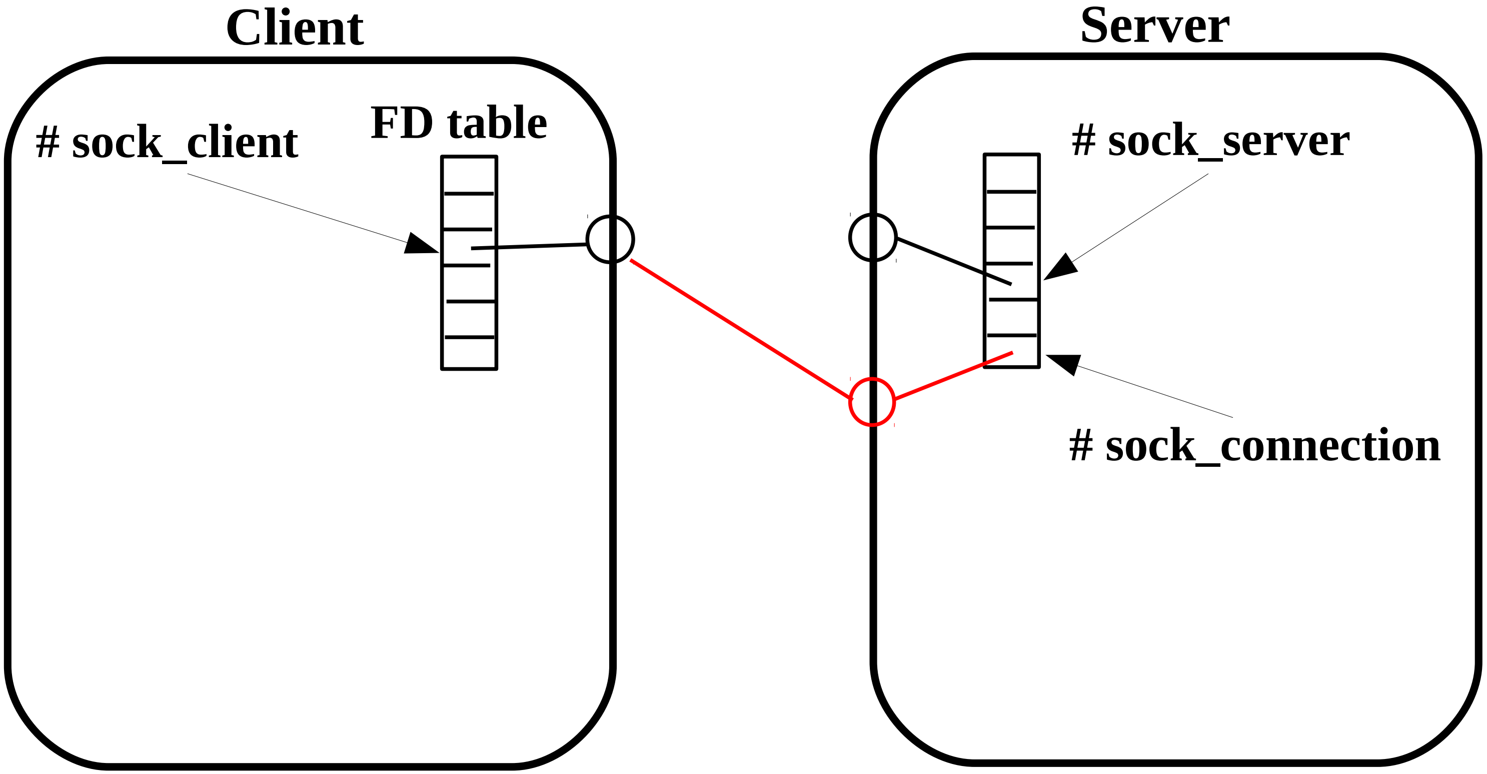
```
int sd;
struct sockaddr_in server; // @IP, #port, mode

sd = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = 0; // let system choose a port
server.sin_addr.s_addr = INADDR_ANY;
                        // any network interface
bind(sd, (struct sockaddr *)&server, sizeof(server));
listen(sd, 5);
```

accept() function

- **int accept(int sock_desc, struct sockaddr *client, int lg_@)**
- **sock_desc: socket descriptor returned by socket()**
- **client: identity of the client which requested the connection**
- **accept returns the socket descriptor associated with the accepted connection**



After call to accept()



Message emission/reception functions

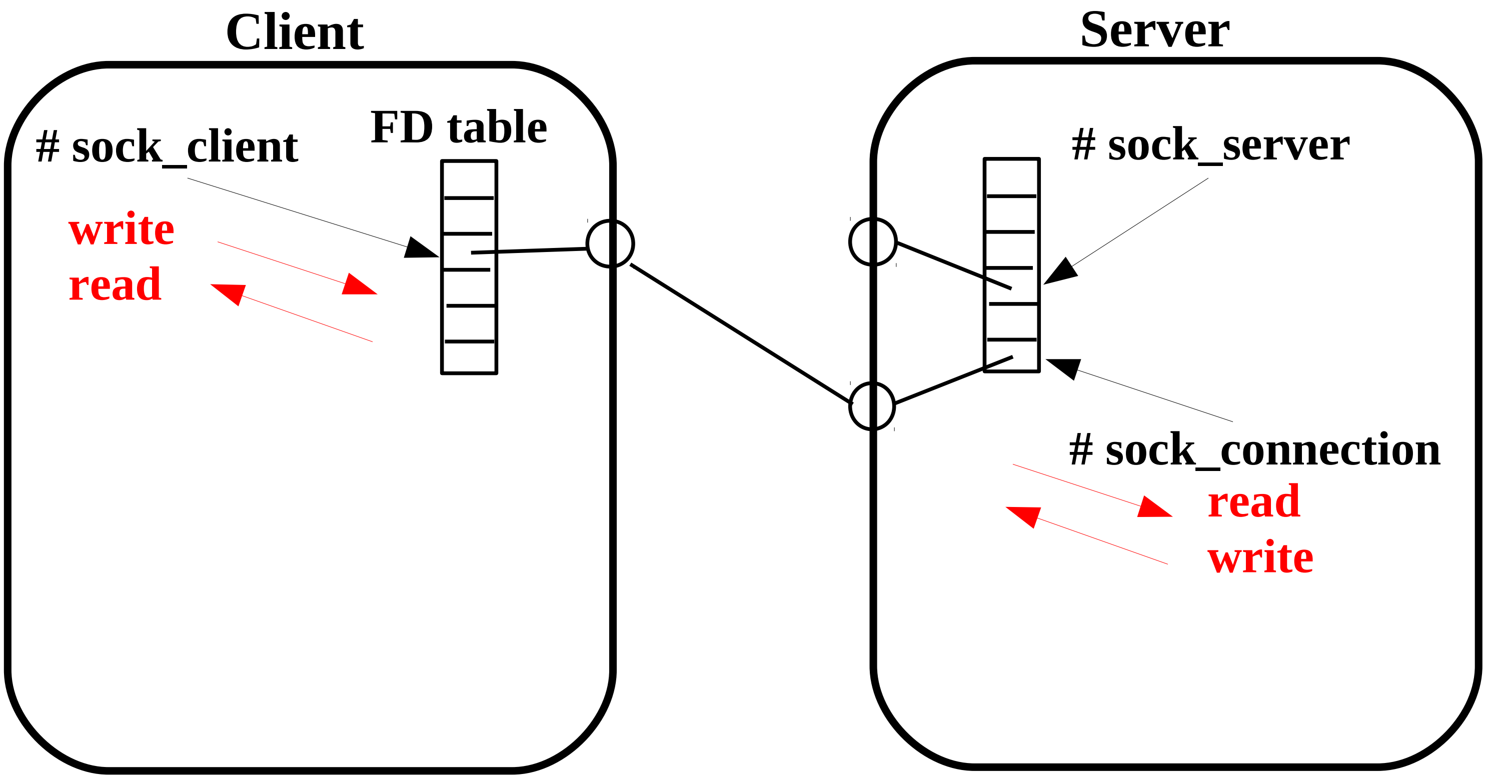
- `int write(int sock_desc, char *buff, int lg_buff);`
- `int read(int sock_desc, char *buff, int lg_buff);`
- `int send(int sock_desc, char *buff, int lg_buff, int flag);`
- `int recv(int sock_desc, char *buff, int lg_buff, int flag);`

-
  ```
  int sendto(int sock_desc, char *buff, int lg_buff, int flag,
             struct sockaddr *to, int lg_to);
  ```

-
  ```
  int recvfrom(int sock_desc, char *buff, int lg_buff, int flag,
             struct sockaddr *from, int lg_from);
  ```

- `flag` : options to control transmission parameters
  (consult man)

Communication



A concurrent server

- After fork() the child inherits the father’s descriptors
- Example of server:

```
int sd, nsd;
...
sd = socket(AF_INET, SOCK_STREAM, 0);
...
bind(sd, (struct sockaddr *)&server, sizeof(server));
listen(sd, 5);
while (!end) {
     nsd = accept(sd, ...);
     if (fork() == 0) {
              close(sd); // the child doesn’t need the father’s socket

              /* here we handle the connection with the client */

              close(nsd); // close the connection with the client
              exit(0); // death of the child
     }
     close(nsd); // the father doesn’t need the socket of the conntion
}
```

Programming Socket in Java

- package **java.net**
  - **InetAddress**
  - **Socket**
  - **ServerSocket**
  - **DatagramSocket / DatagramPacket**

Using **InetAddress** (1)

```
import java.net.*;
public class Enseeiht1 {

  public static void main (String[] args) {
    try {
      InetAddress address =
        InetAddress.getByName("www.enseeiht.fr");
      System.out.println(address);
    } catch (UnknownHostException e) {
      System.out.println("cannot find www.enseeiht.fr");
    }
  }
}
```

Using **InetAddress** (2)

```
import java.net.*;
public class Enseeiht2 {

  public static void main (String[] args) {
    try {
      InetAddress a = InetAddress.getLocalHost();
      System.out.println(a.getHostName() + " / " +
                         a.getHostAddress());
    } catch (UnknownHostException e) {
      System.out.println("No access to my address");
    }
  }
}
```

Client socket and TCP connexion

```
try {
   Socket s = new Socket("www.enseeiht.fr",80);
  …
} catch (UnknownHostException u) {
   System.out.println("Unknown host");
} catch (IOException e) {
   System.out.println("IO exception");
}
```

Reading/writing on a TCP connection

```
try {
    Socket s = new Socket ("www.enseeiht.fr",80);
    InputStream is = s.getInputStream();
    …
    OutputStream os = s.getOutputStream();
    …
} catch (Exception e) {
    System.err.println(e);
}
```

Server socket TCP connection

```
try {
    ServerSocket serveur = new ServerSocket(port);
    Socket s = serveur.accept();
    OutputStream os = s.getOutputStream();
    InputStream is = s.getIntputStream();
    …
} catch (IOException e) {
    System.err.println(e);
}
```

Few words about classes for managing streams

- Suffix: type of stream
  - Stream of bytes (InputStream/OutputStream)
  - Stream of characters (Reader/Writer)
- Prefix: source and destination
  - ByteArray, File, Object …
  - Buffered, LineNumber, …

- https://www.developer.com/java/data/understanding-byte-streams-and-character-streams-in-java.html

Few words about classes for managing streams

	Streams for reading	Streams for writing
Character streams	BufferedReader	BufferedWriter
	CharArrayReader	CharArrayWriter
	FileReader	FileWriter
	InputStreamReader	OutputStreamWriter
	LineNumberReader	
	PipedReader	PipedWriter
	PushbackReader	
	StringReader	StringWriter
Byte streams	BufferedInputStream	BufferedOutputStream
	ByteArrayInputStream	ByteArrayOutputStream
	DataInputStream	DataOuputStream
	FileInputStream	FileOutputStream
	ObjectInputStream	ObjetOutputStream
	PipedInputStream	PipedOutputStream
		PrintStream
	PushbackInputStream	
	SequenceInputStream	

Few words about classes for managing streams

```
BufferedReader br = new BufferedReader(
   new InputStreamReader(socket.getInputStream()));
String s = br.readLine();
```

- InputStreamReader: converts a byte stream into a character stream
- BufferedReader: implements buffering

```
PrintWriter pred = new PrintWriter(
    new BufferedWriter(
          new OutputStreamWriter(
               socket.getOutputStream())));
```

- PrintWriter: formatted printing

Reading on a UDP socket

```
try {
  int p = 9999;
  byte[] t = new byte[10];
  DatagramSocket s = new DatagramSocket(p);
  DatagramPacket d = new DatagramPacket(t,t.length);
  s.receive(d);
  String str = new String(d.getData(), 0, t.length);
  System.out.println(d.getAddress()+"/"+d.getPort()+"/"+str);
  …
}
catch (Exception e) {
  System.err.println(e);
}
```

sender

Writing on a UDP socket

```
try {
    int p = 8888; // for receiving a response
    byte[] t = new byte[10];
    FileInputStream f = new FileInputStream("data.txt");
    f.read(t);
    DatagramSocket s = new DatagramSocket(p);
    DatagramPacket d = new DatagramPacket(t, t.length,
      InetAddress.getByName("thor.enseeiht.fr"), 9999);
    s.send(d);
    …
} catch (Exception e) {
    System.err.println(e);
}
```

destination

A full example:
TCP + serialization + threads

Passing an object (by value) with serialization

The object to be passed:

```
public class Person implements Serializable {
    String firstname;
    String lastname;
    int age ;
    public Person(String firstname, String lastname, int age) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.age = age;
    }
    public String toString() {
        return this.firstname+" "+this.lastname+" "+this.age;
    }
}
```

A full example:
TCP + serialization + threads

The client

```
public class Client {
   public static void main (String[] str) {
      try {
         Socket csock = new Socket("localhost",9999);
         ObjectOutputStream oos = new ObjectOutputStream (
                                     csock.getOutputStream());
         oos.writeObject(new Person("Dan","Hagi",53));
         csock.close();
      } catch (Exception e) {
                 System.out.println("An error has occurred ...");
      }
   }
}
```

A full example:
TCP + serialization + threads

The server

```
public class Server {
   public static void main (String[] str) {
      try {
          ServerSocket ss;
          int port = 9999;
          ss = new ServerSocket(port);
          System.out.println("Server ready ...");
          while (true) {
             Slave sl = new Slave(ss.accept());
             sl.start();
          }
       } catch (Exception e) {
          System.out.println("An error has occurred ...");
       }
   }
}
```

A full example:
TCP + serialization + threads

The slave

```
public class Slave extends Thread {
   Socket ssock;
   public Slave(Socket s) {
      this.ssock = s;
   }
   public void run() {
      try {
          ObjectInputStream ois = new ObjectInputStream(
                                    ssock.getInputStream());
          Person v = (Person)ois.readObject();
          System.out.println("Received person: "+ v.toString());
          ssock.close();
      } catch (Exception e) {
          System.out.println("An error has occurred ...");
      }
   }
}
```

Conclusion

- Programming with sockets
  - Quite simple
  - Allow fine-grained control over exchanges messages
  - Basic, can be verbose and error prone
- Higher level paradigms
  - Remote procedure/method invocation
  - Message oriented middleware / persistent messages
  - ….

Many tutorials about socket programming on the Web …

Example : https://www.tutorialspoint.com/java/java_networking.htm