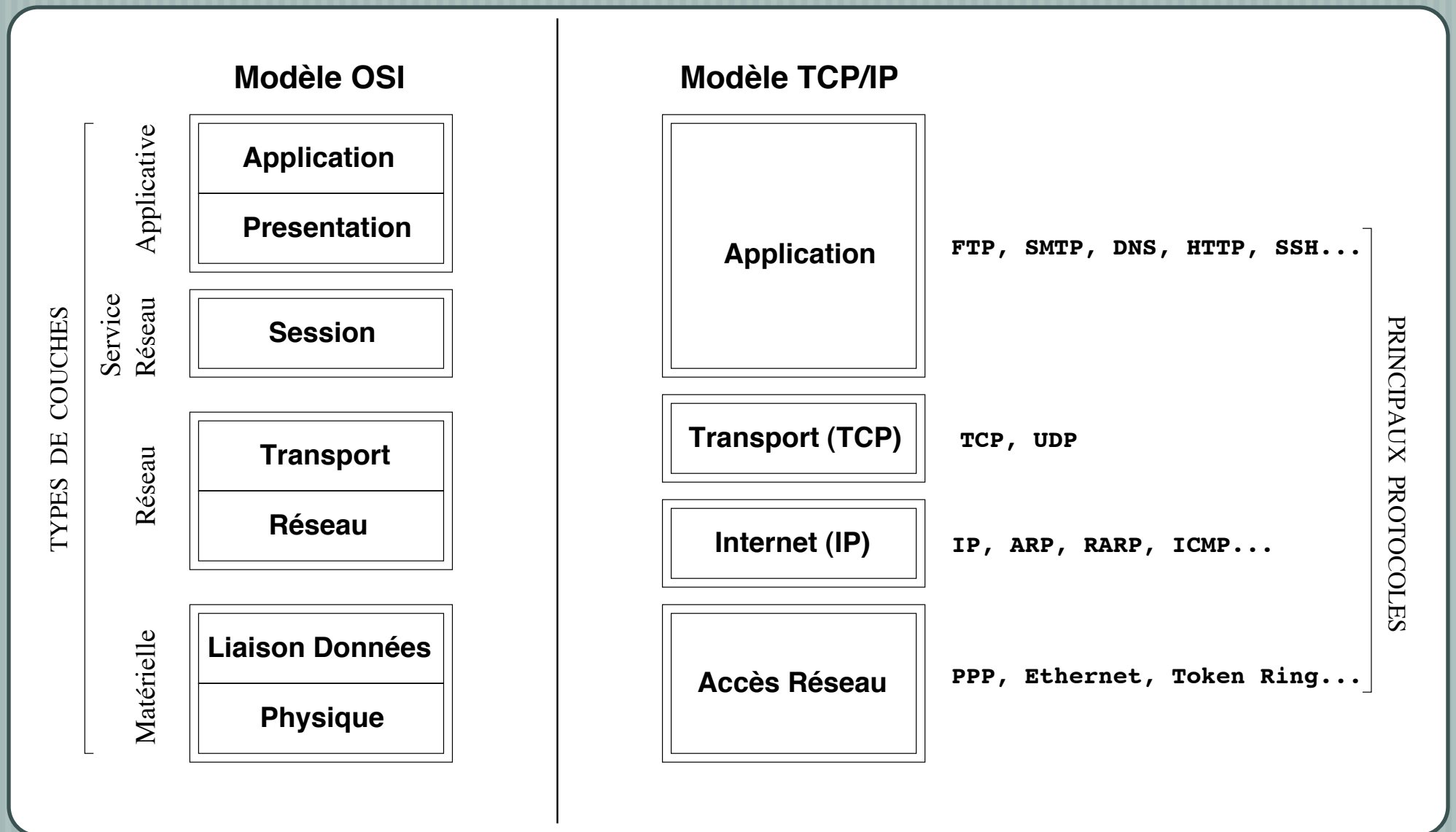
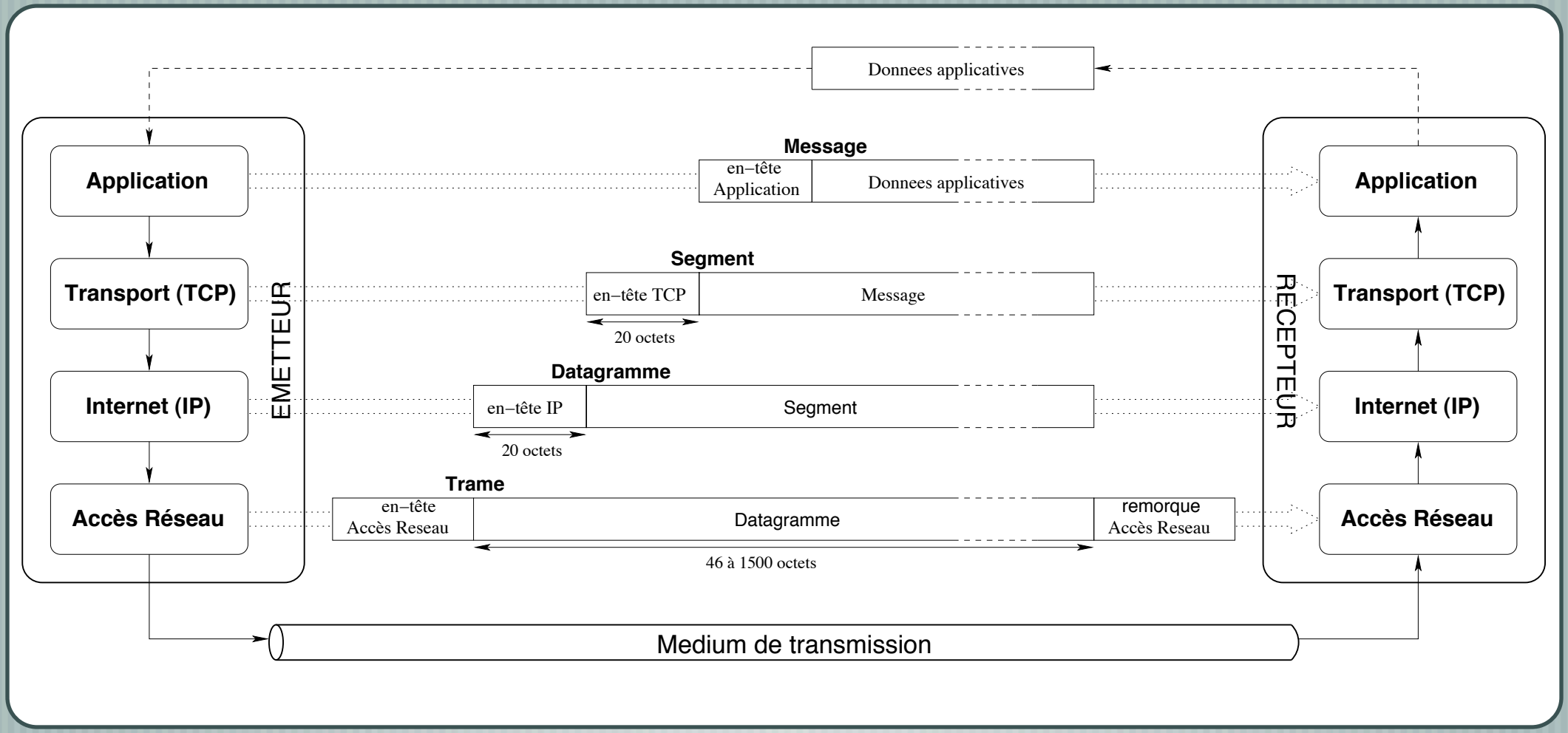
Socket Programming

(Dr. ☺) Sébastien Varrette

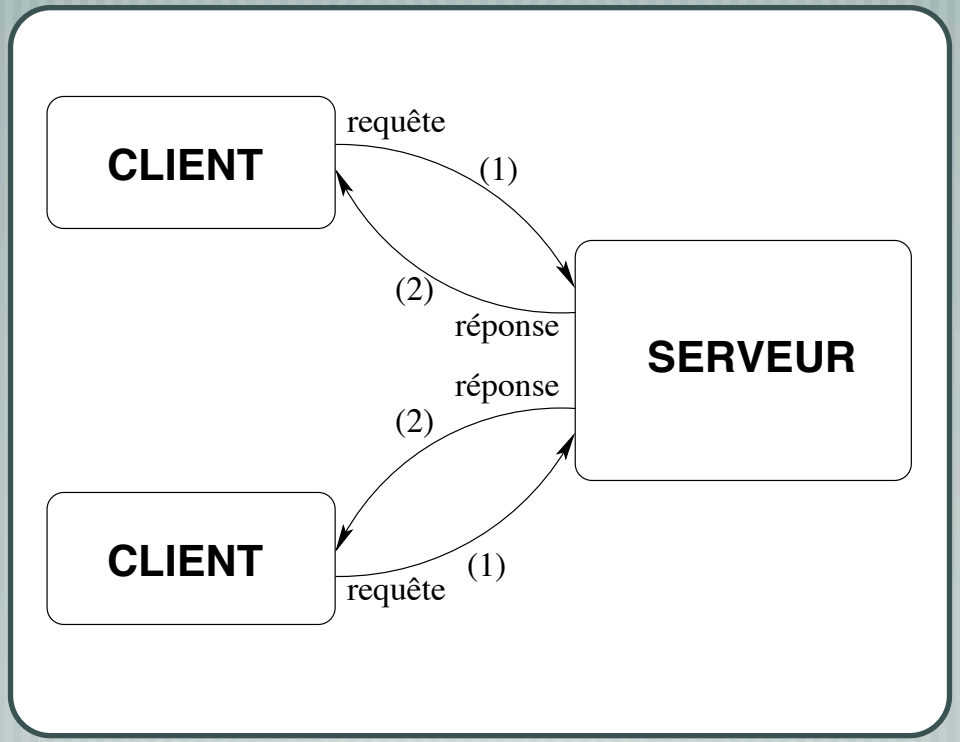
Network Model



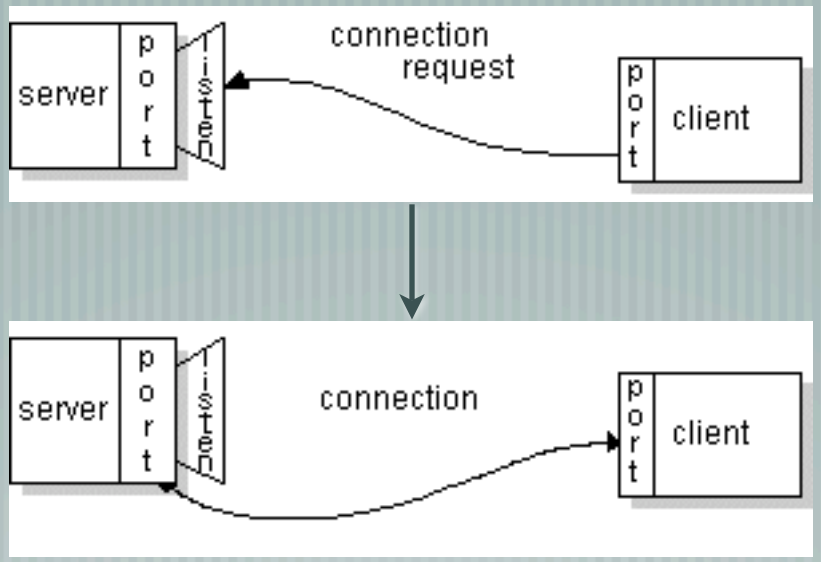
Data Encapsulation



Client/Server Model



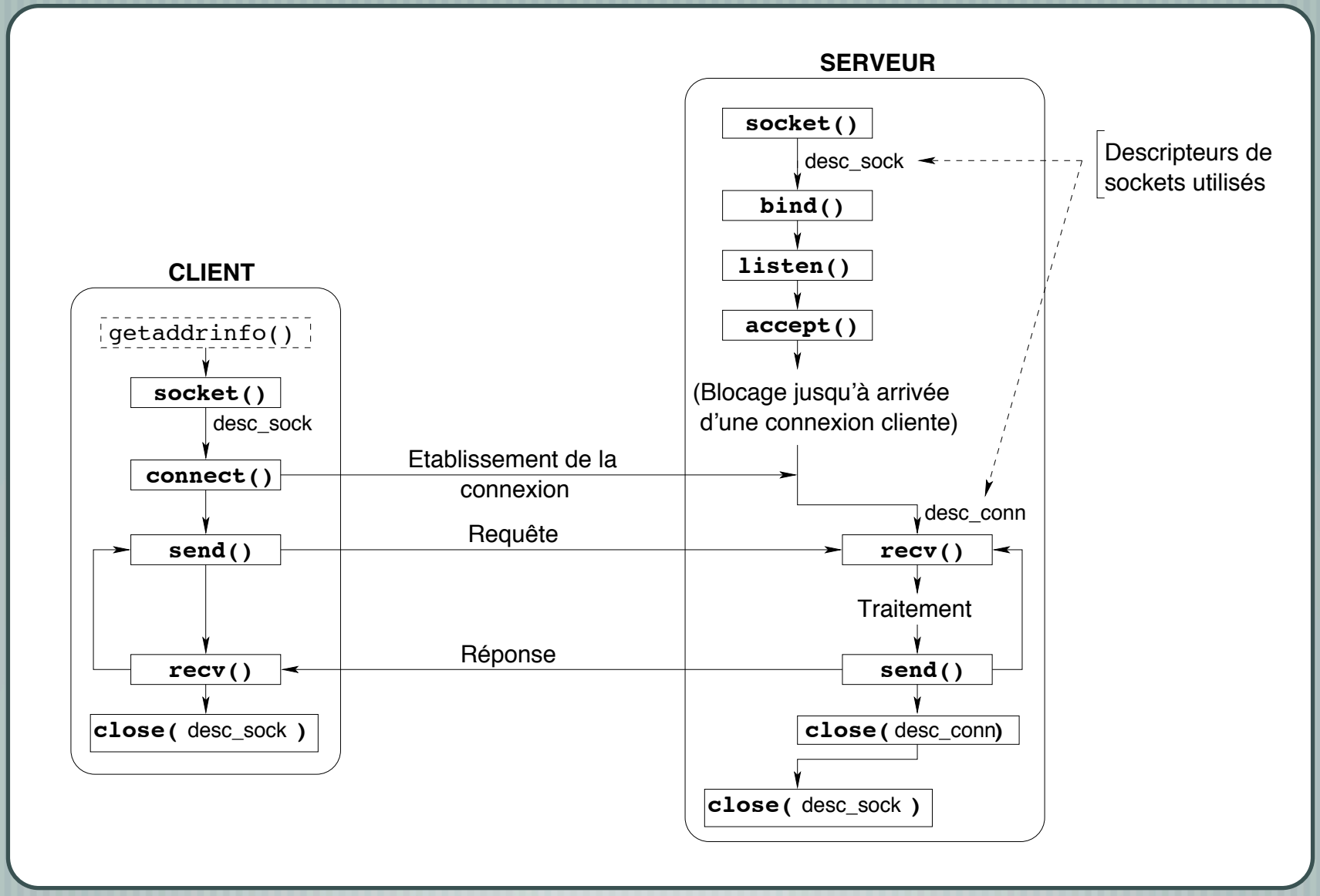
Sockets in TCP connection



Definition: A *socket* is one endpoint of a two-way communication link between two programs running on the network.

- TCP Socket = <@IP,port> = <InetAddress,int> (in Java)
- TCP connection = <EndPoint1, EndPoint2>

TCP sockets in C



TCP sockets in Java

`java.net` package:

- Socket class (Client)

`Socket(InetAddress address, int port)`	Open a TCP connection and create the associated Socket object
`void close()`	Close this socket
`OutputStream getOutputStream()`	Returns an output stream for this socket.
`InputStream getInputStream()`	Returns an input stream for this socket.

- SocketServer class (Server)

`ServerSocket(int port)`	Creates a server socket, bound to the specified port.
`Socket accept()`	Listens for a connection to be made to this socket and accepts it.

Socket client: basic scheme

1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server's protocol.
4. Close the streams.
5. Close the socket.

Sockets - client

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

// .....

int    port   = 80;
Socket client = null;

try {
    InetAddress addr = InetAddress.getByName("myserver.uni.lu");
    client = new Socket(addr, port); // block until connect succeeds
}
catch (UnknownHostException e) { } // eventually from getByName()
catch (IOException e)          { } // eventually from Socket()
finally {
    if (client != null) client.close();
}
```

Sockets - server

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

// .....

int          port   = 666;
ServerSocket server = null;
try {
    server = new ServerSocket(port);
    /**
     * listen for new connection requests.When a request arrives,
     * service it and resume listening for more requests.
     */
    while (true) {
        // now listen for connections
        Socket client = sock.accept();
        ServiceConnectionOn(client);
    }
}
catch (IOException e) { }
finally {
    if (server != null) server.close();
}
```

Sockets - I/O

- Data exchanged: text/binaries/serialized
  - ✓ interact with `getInputStream()` and `getOutputStream()`
  - ✓ Eventually convert bytes → char stream
    - use `InputStreamReader` and `OutputStreamWriter`
- Text read : use `BufferedReader`
- Text write : use ~~`BufferedWriter`~~ `PrintWriter`
  - ✓ access to print(), println()...

Sockets - Read

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

// .....

BufferedReader input = null;

try {
    input = new BufferedReader(
                    new InputStreamReader(socket.getInputStream())
    );
    String str;
    while ((str = input.readLine()) != null) {
        process(str); // do your job on the received line
    }
}
catch (IOException e)           { }
finally {
    if (input != null) input.close();
}
```

Sockets - Write

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

// .....

PrintWriter output = null;

try {
    // Create a new PrintWriter, with automatic line flushing
    output = new PrintWriter(sock.getOutputStream(),true);
    output.println("Hello!");
}
catch (IOException e)           { }
finally {
    if (output != null) output.close();
}
```

Exercise 1: HTTP client

- minimal HTTP request [RFC 1945]:
  - carriage return: “\r\n”
  - first ‘/index.html’ : path to requested file

```
GET /index.html

```

Implement an HTTP client that :

1. connect by TCP to an HTTP server
2. ask for an HTML page
3. print on stdout the received HTML text

Exercise 2: echo C/S

- Client send a message to the server through sockets
- Server answer to the client by echoing the received message
  - Exercise 2(a) : command line version (quit on “bye”)
  - Exercise 2(b) : swing version of the client
  - Exercise 2(c) : multithreaded server version
    - use the Runnable interface

Exercise 3: Following a Protocol

- Implement a C/S respecting a protocol, for instance:

Server: "Knock knock!"
Client: "Who's there?"
Server: "Dexter."
Client: "Dexter who?"
Server: "Dexter halls with boughs of holly."
Client: "Groan."

  - Client class : `KnockKnockClient`
  - Server :
    - Class : `KnockKnockServer`
    - Protocol definition class: `KnockKnockProtocol`
      (Keep track of the current state)

Appendix

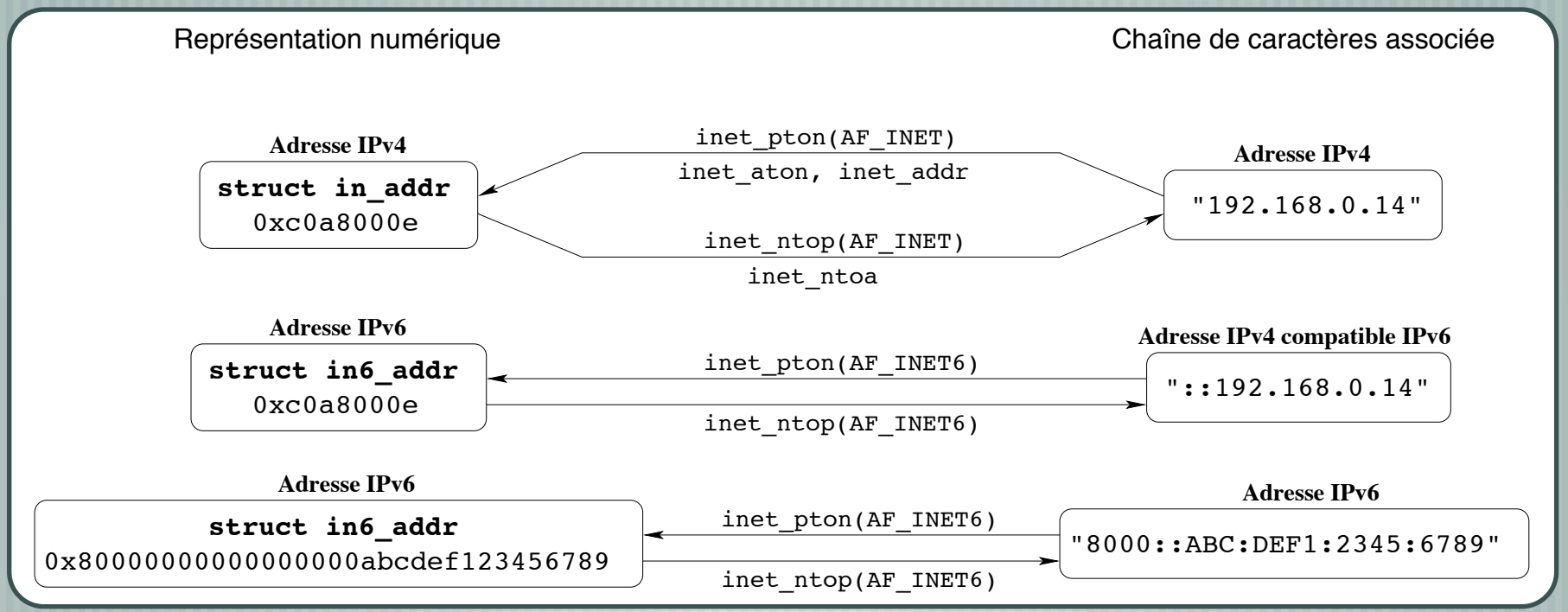
IP address Manipulations in C

IPv4 :

```
typedef uint32_t    in_addr_t ;
struct  in_addr {           /* Adresse IP sur 32 bits (IPv4) */
        in_addr_t  s_addr;  /* (dans l'ordre des octets du réseau). */
};
```

IPv6:

```
struct  in6_addr {                  /* Adresse IP sur 128 bits (IPv6) */
          uint8_t     s6_addr[16];  /* (dans l'ordre des octets du réseau). */
};
```



Exercice 2(a)

Command line version (quit on “bye”)

FalkorMac Terminal — bash — 81x9

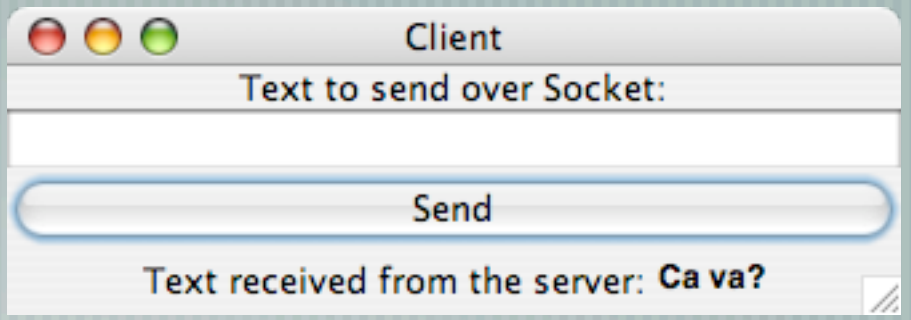
```
[12:09:47] sebastien.varrette@falkor Exo2-Echo> java EchoClient localhost 6666
salut
[Answer from server]: salut
ca va?
[Answer from server]: ca va?
t'es lourd de tout répéter
[Answer from server]: t'es lourd de tout répéter
bye
[12:10:06] sebastien.varrette@falkor Exo2-Echo>
```

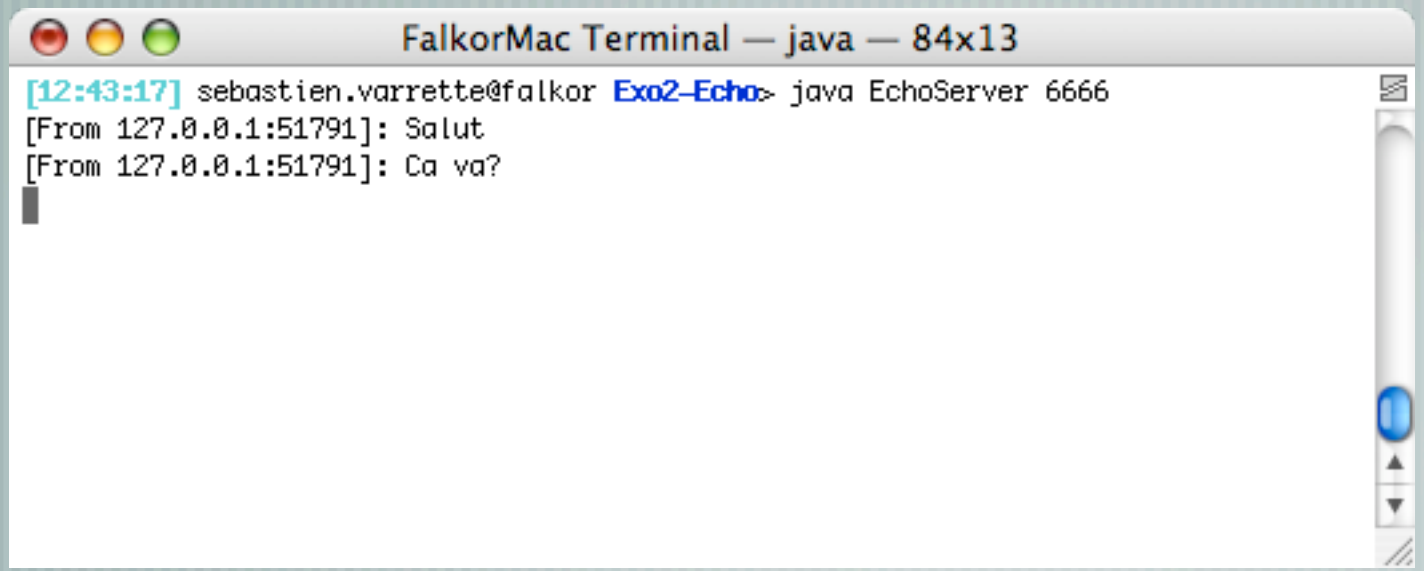
FalkorMac Terminal — java — 69x9

```
[12:09:36] sebastien.varrette@falkor Exo2-Echo> java EchoServer 6666
[From 127.0.0.1:51048]: salut
[From 127.0.0.1:51048]: ca va?
[From 127.0.0.1:51048]: t'es lourd de tout répéter
Close connection to 127.0.0.1:51048
```

Exercice 2(b)

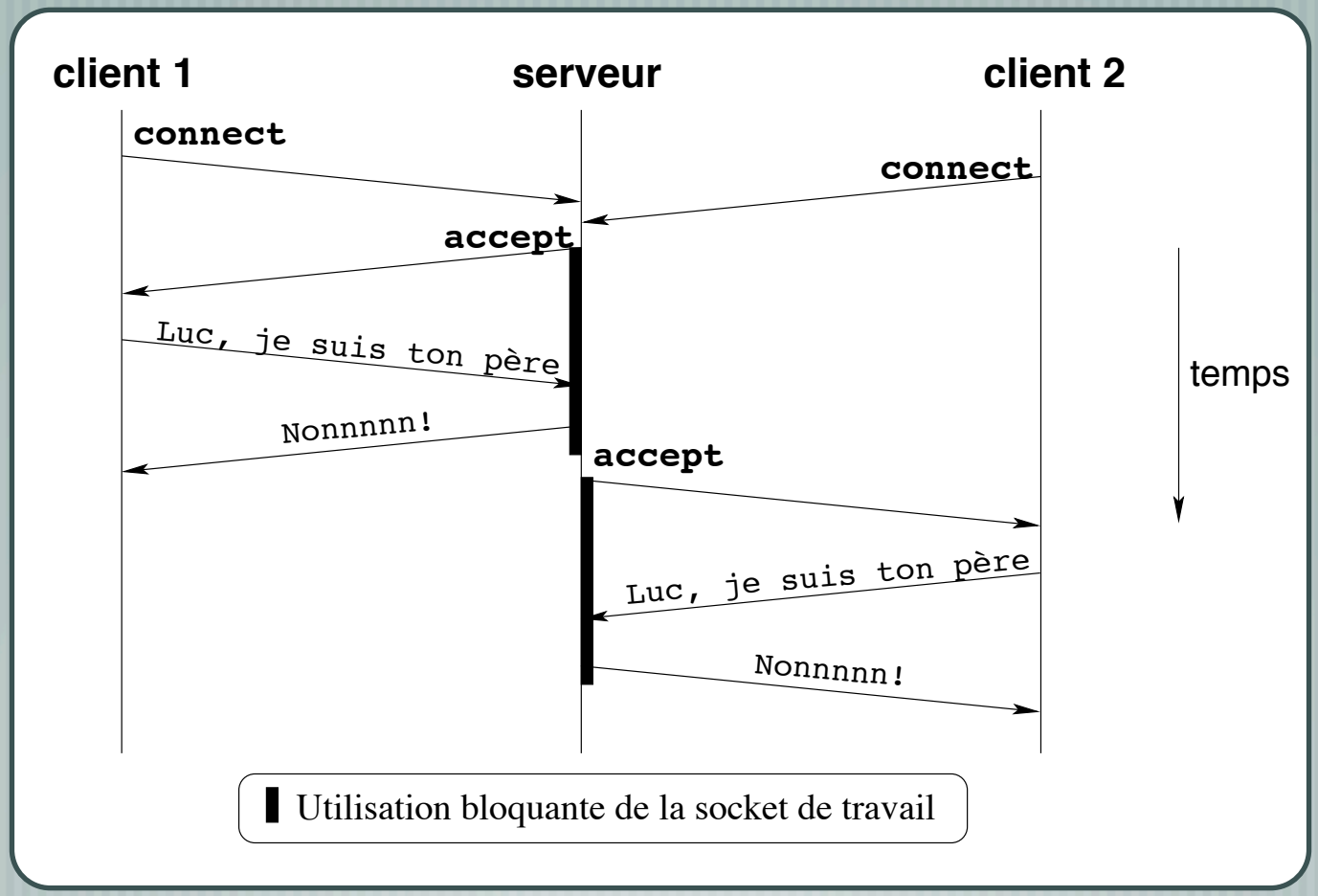
- Swing version





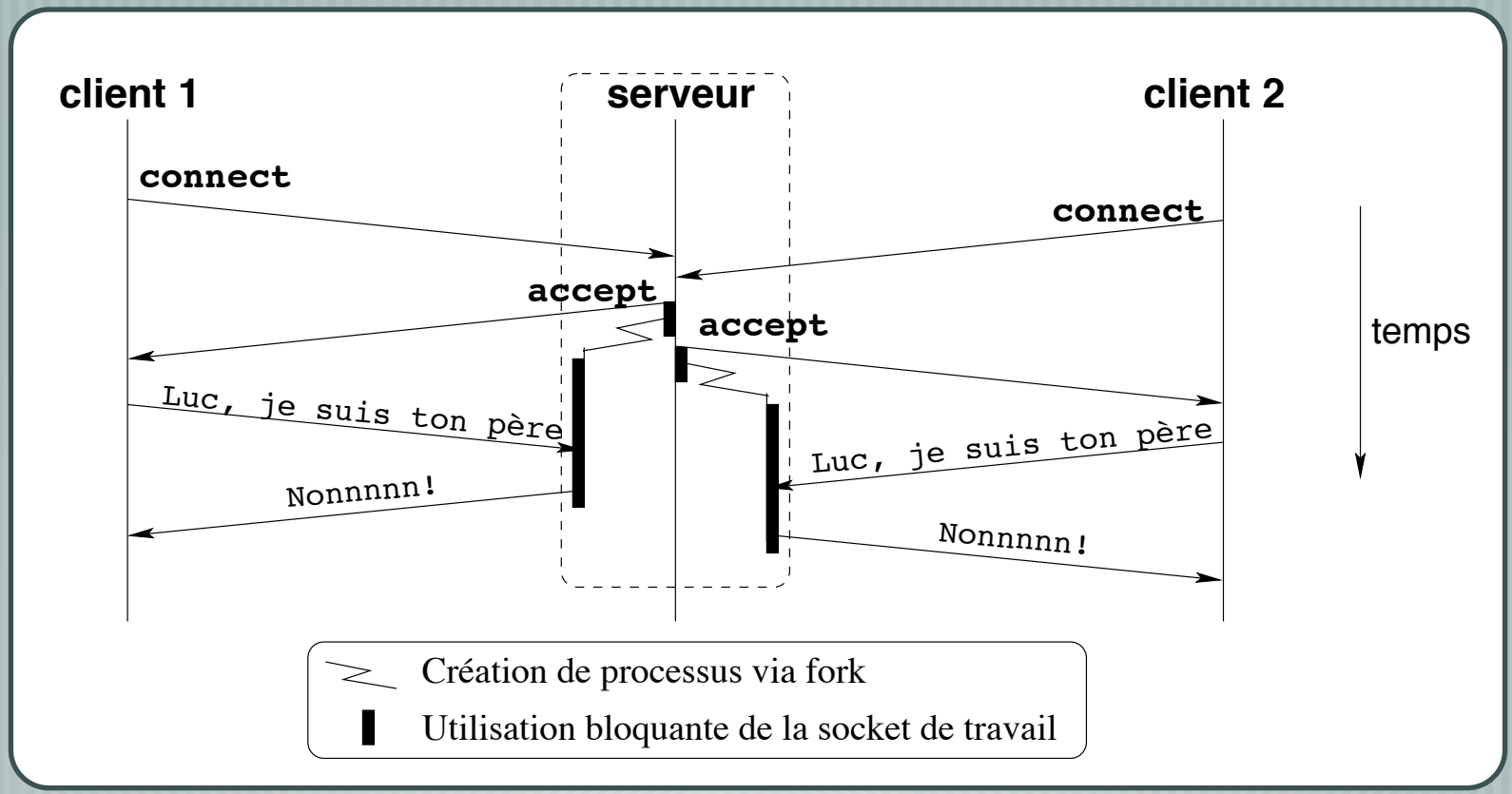
Concurrent Access (in C)

Basic behaviour:

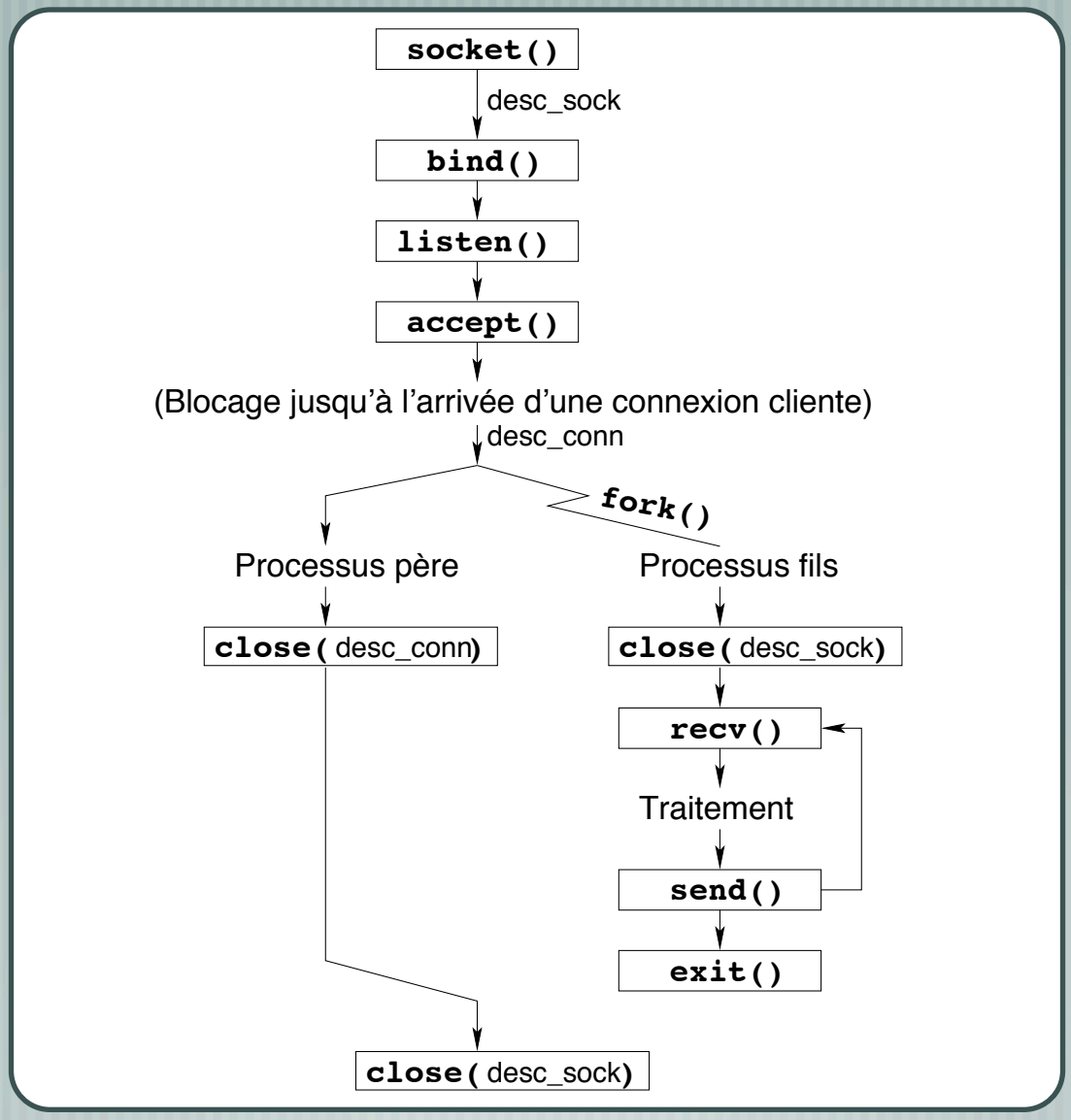


Concurrent Access (in C)

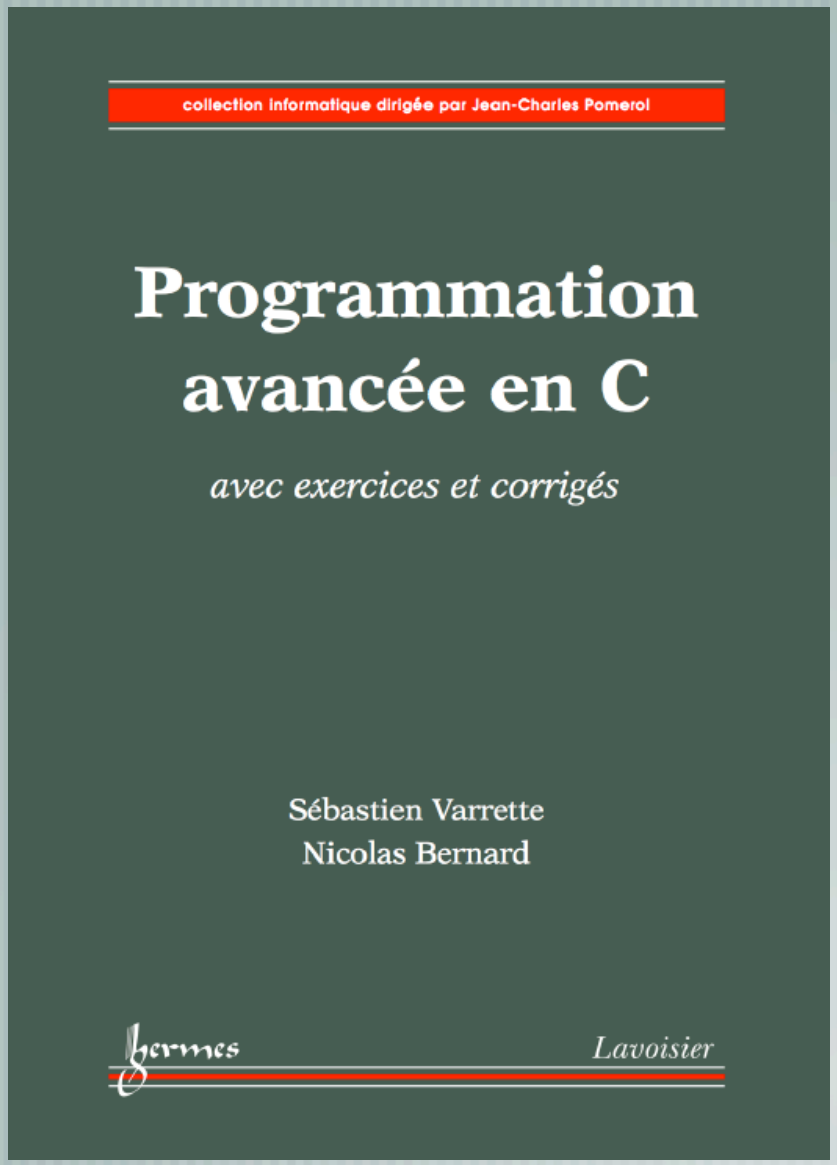
Expected behaviour:



TCP socket (Server) in C



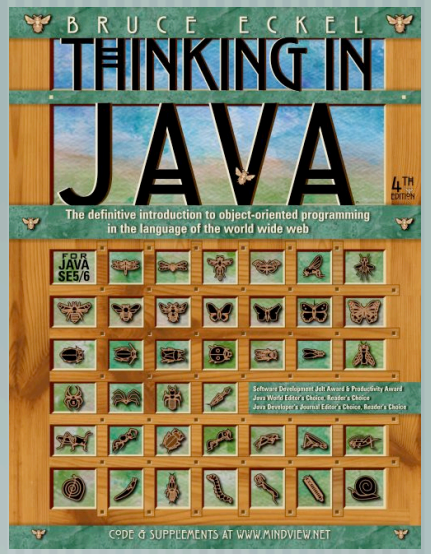
More infos in C



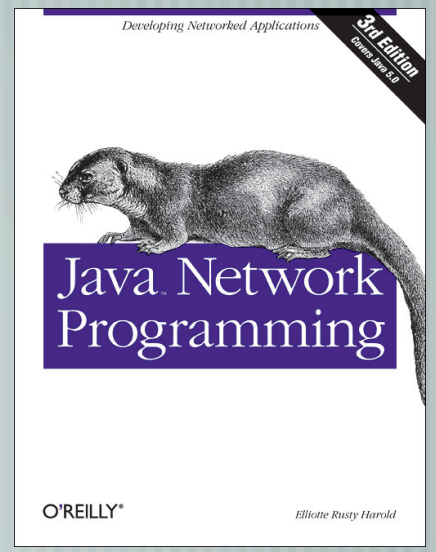
cf. http://c.lafraze.net

More info for Java

- Thinking in Java: online Available!
  - http://mindview.net/Books



- The best tutorials: http://java.sun.com



- Java Network Programming