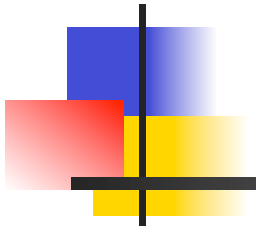
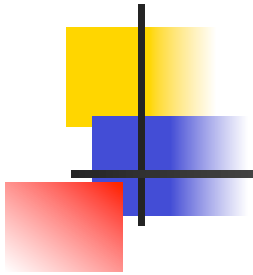


Network Programming

L. Pautet

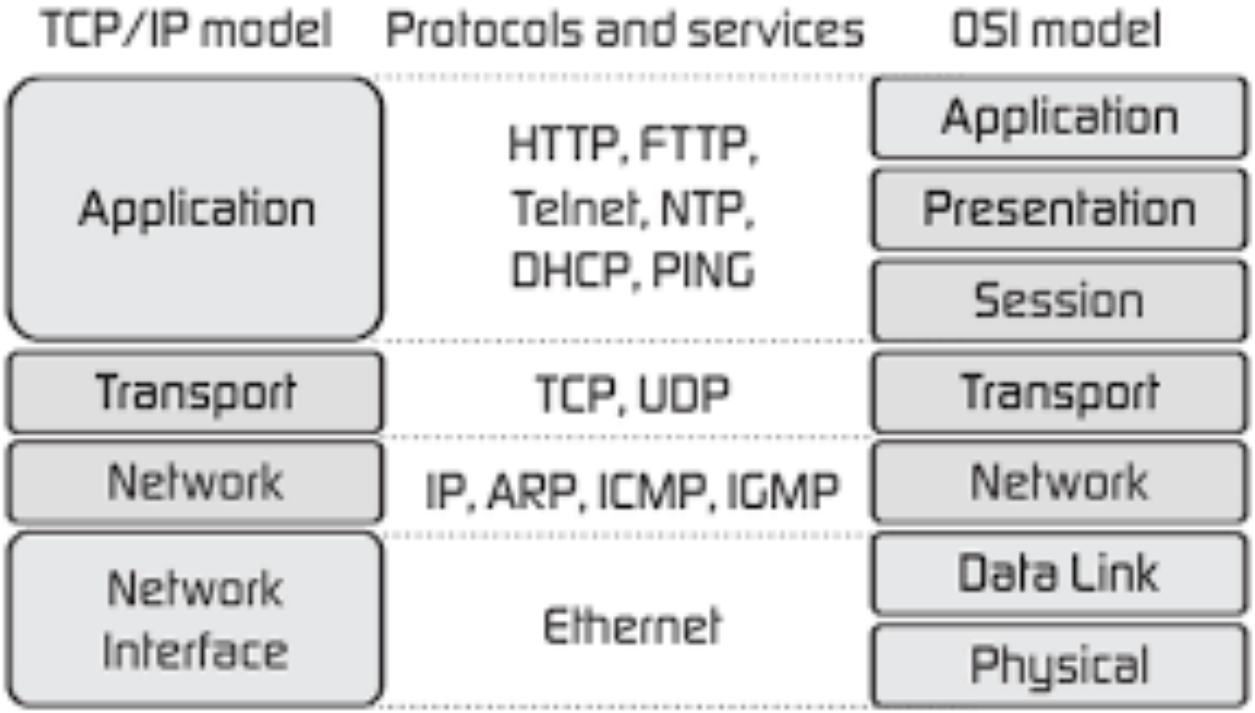
Version EN, 2.0

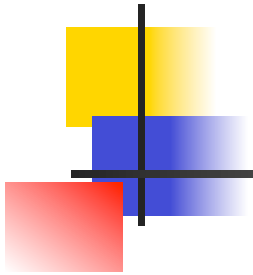


IP, UDP et TCP

- IETF has defined a series of protocols:
  - IPv4 and IPv6 for addressing machines;
  - TCP and UDP for exchanging messages over IP:
  - These are basic standards from a user point of view
  - Other protocols concern physical or application aspects
  - The IETF only defines a message format and the controllers for establishing connections, exchanging messages, handling errors, fragmentation, etc.

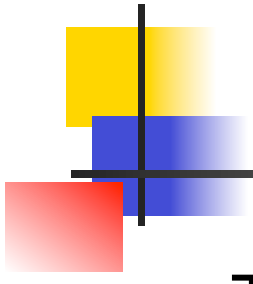
OSI Model vs TCP/IP Model





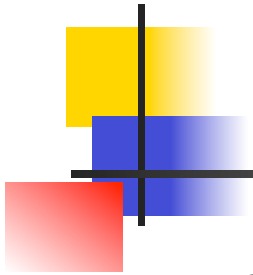
IPv4 address space

- IPv4 uses 32_bit addresses with a quad-dotted decomposition:
  - 137.194.2.34, netid = 137.194, hostid = 2.34
- An IPv4 address is divided into two parts: netid et hostid
  - Netid : network identifier
  - Hostid : host identifier
- Used for routing and network interface identification
- IPv6 was developed by the IETF to deal with IPv4 address exhaustion. Supposed to replace IPv4 but the move is complex.



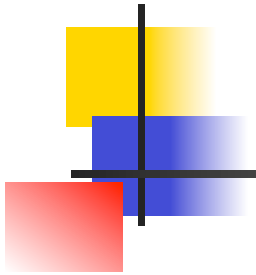
IPv4 address space

- The netid is partitioned into network classes:
  - Class A, netid coded on 1 byte (leading bits 0):
    - Addresses from 1.0.0.0 to 126.0.0.0 (127 : specific to localhost)
  - Class B, netid coded on 2 bytes (leading bits 10) :
    - Addresses from 128.0.0.0 to 191.255.0.0
  - Class C, netid coded on 3 bytes (leading bits 110) :
    - Addresses from 192.0.0.0 to 223.255.255.0
  - Class D (multicast), netid on 3 bytes (leading bits 1110)
    - Addresses from 224.0.0.0 to 239. 255.255.0



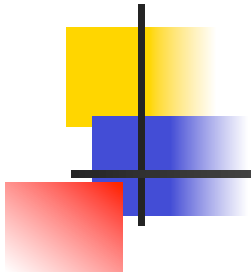
IPv4 address space

- Specific addresses
  - 127.0.0.1 : « localhost », loopback address
  - 0.0.0.0 : invalid address
- Reserved private IPv4 addresses :
  - Class A : netid 10, hostid 0.0.1 to 255.255.255
  - Class B : netid 172.16.0 to 172.31, hostid 0.0 to 255.255
  - Class C : netid 192.168 to 192.168, hostid0.0 to 255.255



TCP vs UDP

- TCP: connection oriented protocol above IP
  - Reliable protocol (with error handling)
    - Maximum packet size (MTU), with a segmentation mechanism for large data
  - Flow management mechanism to avoid saturating the network (Nagle algo)
- UDP: protocol simpler than TCP
  - If "everything is fine » (LAN), we avoid the complexity of TCP
    - We lose the sequencing of the packets, the flow management
    - Useful for light applications, soft real time (multimedia)

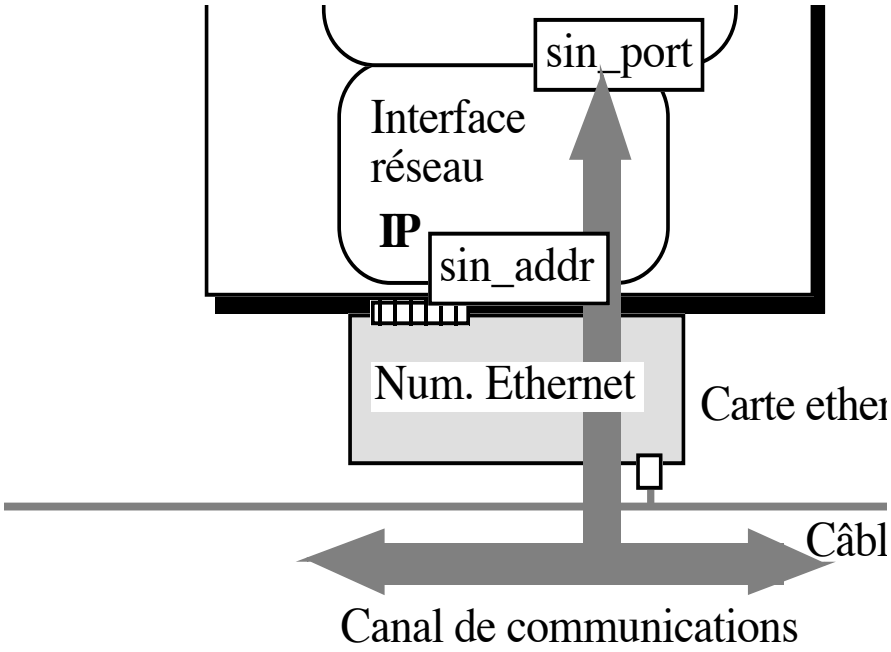


BSD Sockets
An API for IP, TCP and UDP

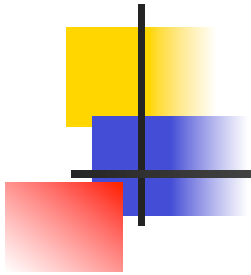
- No API defined by the IETF
- BSD Sockets: model used in Unix
  - Adapted for other platforms, including Windows
- Inspired by the Unix resource model:
  - Every resource is file
  - Sockets follow producer / consumer semantics and are used with open / read / write
  - Strong links with the standard API for the connection lifecycle

Address Family: AF_INET

- AF_INET is used to identify IPv4 internet address family :

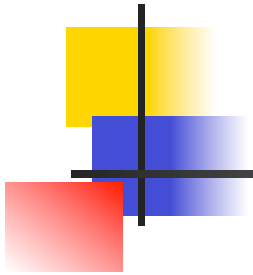


Connection-Oriented Protocol TCP on client side



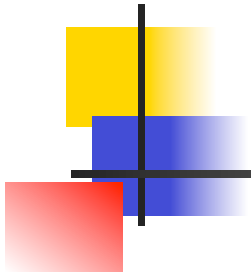
- The caller (client calling a service of a server):
  - Create a socket
  - Build the network address (IP address + port)
    - The server is identified by its IP address or its name
    - The service is identified on the server by its port
    - No name directory for ports, only for IP addresses (DNS)
  - Connect to the server (with the previous address) (automatic assignment of a port to the client)
  - Read from or write to the socket
  - Close the socket

Connection-Oriented Protocol TCP on server side



- The callee (or server and its service):
  - Create a socket
  - Associate a network address (IP address + port) to the socket (and to the service)
  - Wais for incoming connections requests
  - For each incoming connection:
    - Accept the connection (a new socket is created);
    - Read from or write to the new socket
    - Close the new socket

Canvas of a TCP client
C API

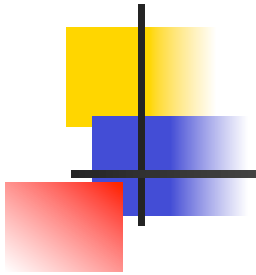


1. Build the server address
2. Directory for host but not for port
    1. gethostbyname is blocking (request to a predefined name server)
3. Request a connection with server

```
Socket = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr ServerAddr;
ServerAddr.sin_addr = gethostbyname(« www.enst.fr »);
ServerAddr.sin_port = ServerPort;
connect(Socket, &ServerAddr, sizeof(struct sockaddr));
```

Canvas of a TCP server
C API in a concurrent system

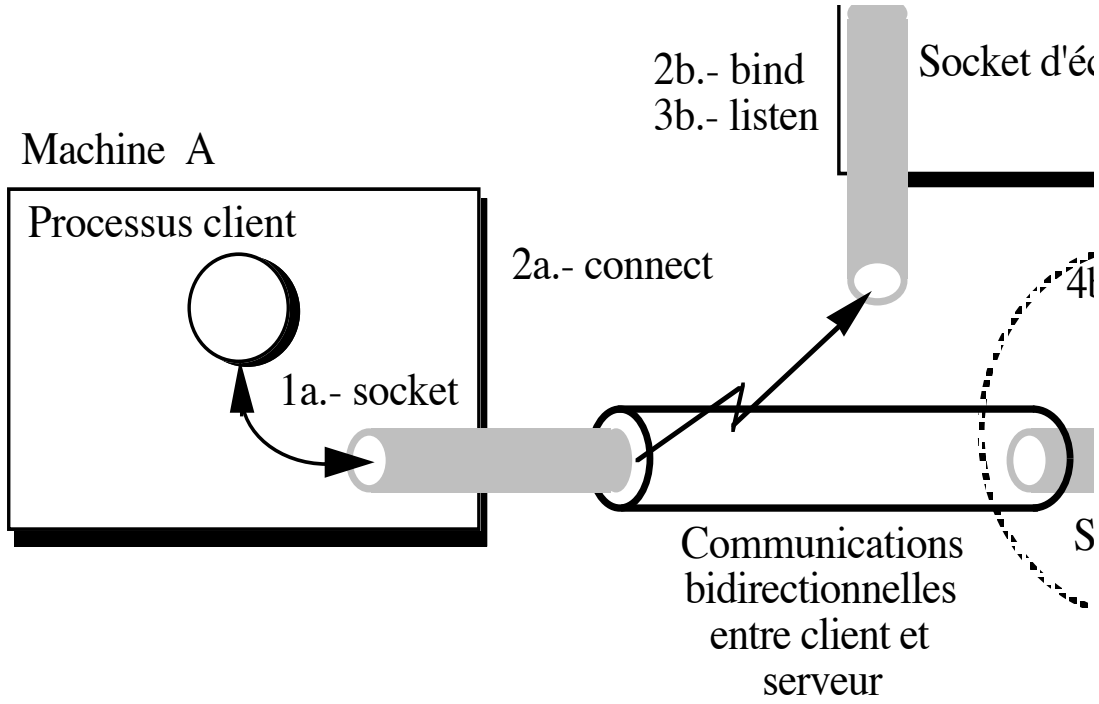
```
struct sockaddr ServerAddr, ClientAddr;
int ClientAddrLen;
ServerSocket = socket(AF_INET, SOCK_STREAM, 0);
bind(ServerSocket, &ServerAddr, sizeof(struct sockaddr));
while (1){ // loop waiting for incoming connnection requests
   Socket = accept(ServerSocket, &ClientAddr, &ClientAddrLen);
   if (fork() == 0){
      close(ServerSocket);
      Handle(Socket);
   } else
      close(Socket);
}
```



Canvas of a TCP server C API in a concurrent system

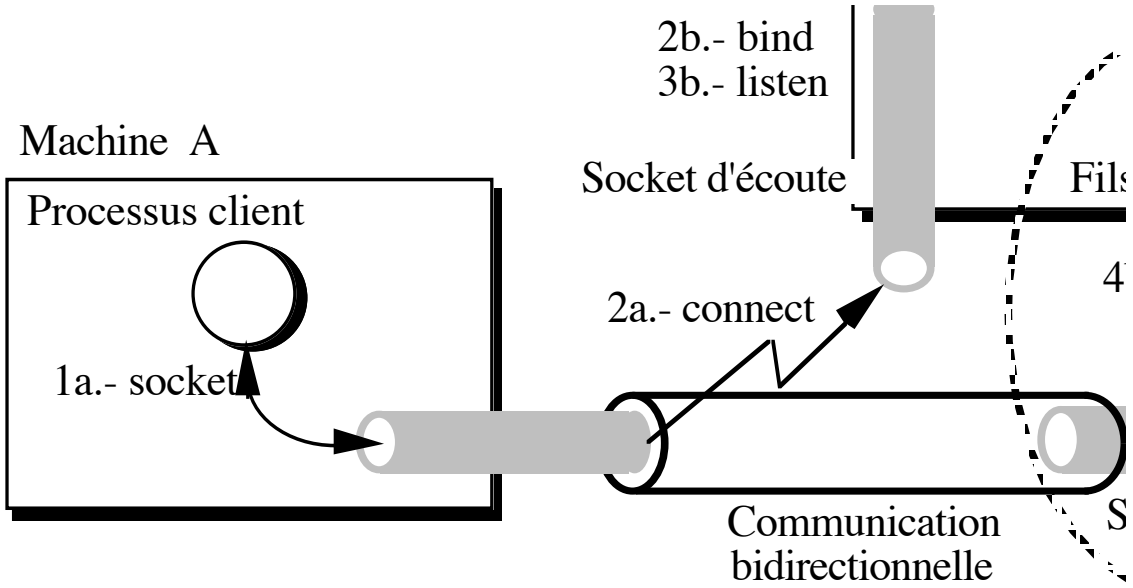
- Create a listening socket (server socket)
- Wait for a connection request (from a client)
- The listening socket creates a communication socket when receiving a client request
- Concurrent management
  - 2 sockets (listening socket + communication socket)
  - 2 processes (or threads) => Use of patterns
    - Leader / Followers (leader accepts & delegates to followers)
    - Half/Sync – Half/Async (accept, connect, read, write block)
    - Executor service

TCP
Overview



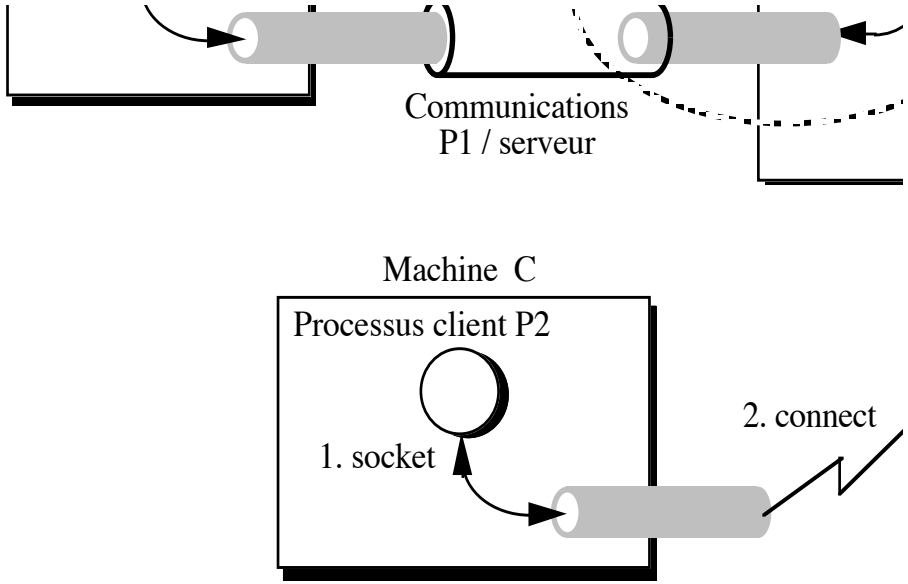
TCP
Concurrent Management (1/2)

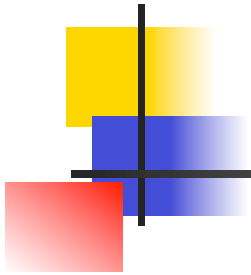
- Communication with a first client



TCP
Concurrent Management (2/2)

- Communication with a second client

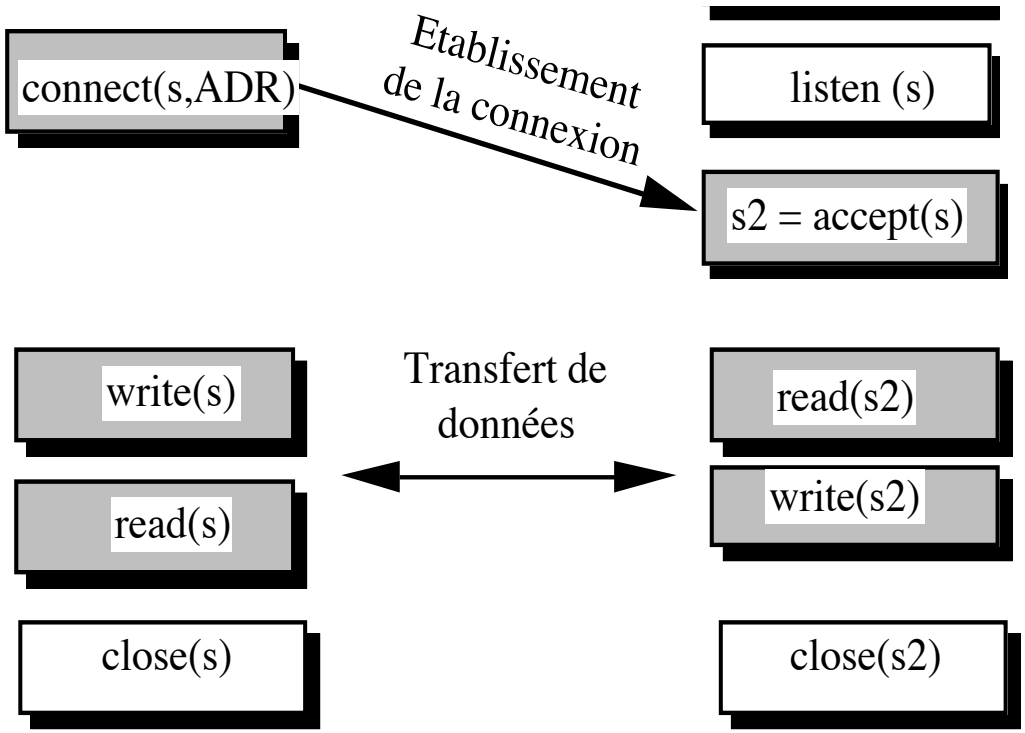


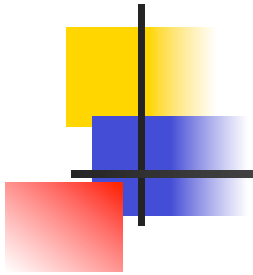


TCP
Socket creation

- Socket = socket(Domain, Type, Protocol)
  - Socket: index from open file table (per process)
  - Domain = AF_INET or PF_INET
  - Type = SOCK_STREAM (TCP)
- bind(Socket, &ServerAddr, ServerAddrLen);
  - To bind socket to a network interface & a port:
  - Socket socket id returned by socket()
  - ServerAddr structure including address and port
  - ServerAddrLen size of structure (sizeof)

TCP
Blocking operations

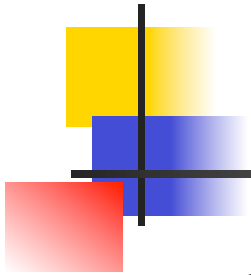




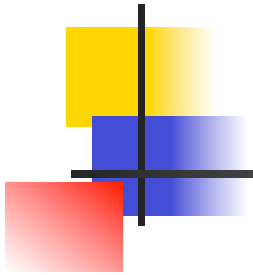
TCP
Socket on client side

- On the client side
  - connect(Socket, &ServerAddr, ServerAddrLen)
  - Initiate a connection on a socket
    - This call attempts to make a connection to another socket on the server side. The other socket is specified by address, which is an address and a port.

Connection-Oriented Protocol Socket on server side



- On the server side
  - listen(ServerSocket, Nb_Clients)
    - maximum length for the queue of pending connections
  - ClientSocket = accept(ServerSocket, &ClientAddr, &ClientAddrLen)
  - Extract the first connection request on the queue of pending connections and create a new socket ClientSocket with same properties of ServerSocket
    - ClientAddr is filled in with the address of the client, the format is determined by the domain in which the communication is occurring.



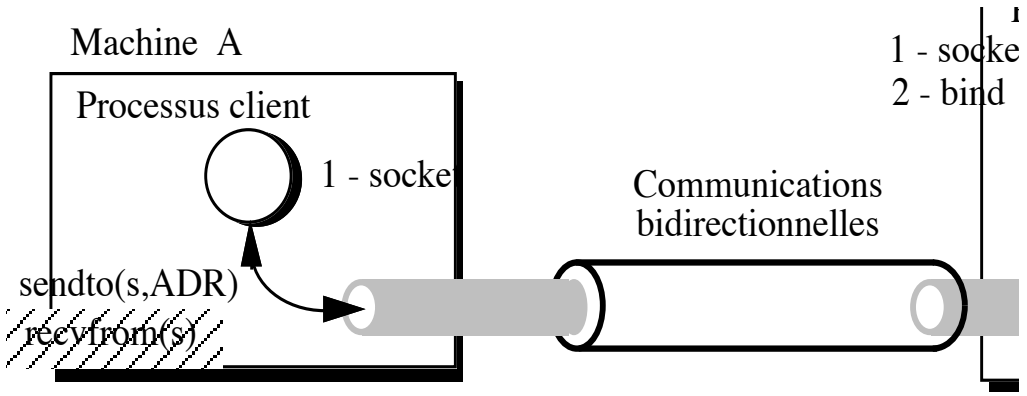
TCP
Sending and receiving messages

1. Standard functions for files:
   - read/write(Socket, Message, MessageLen)

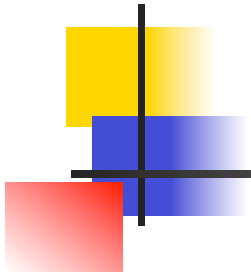
2. Specific function (fine grain control):
   - send/recv(Sock, Message, MessageLen, Option)
     - Example of options : MSG_PEEK
     - Peeks at an incoming message. The data is treated as unread and the next recv() or similar function shall still return this data

Connectionless communication UDP

- Client (caller) :
  - Create a socket ;
  - Read or write on socket ;
- Server (callee) :
  - Create a socket ;
  - Bind socket to an address (IP + port)
  - Read or write on socket



Connectionless communication UDP



- Send a message through connectionless-mode socket (or connection-mode but address ignored)
  - sendto(Socket, Message, MessageLen, 0, &ReceiverAddr, ReceiverAddrLen)
- Receive a message from connectionless-mode socket (or connection-mode but useless)
  - recvfrom(Socket, Message, MessageLen, Flags, &SenderAddr, &SenderAddrLen)
  - SenderAddr permits the application to retrieve the source address of received data

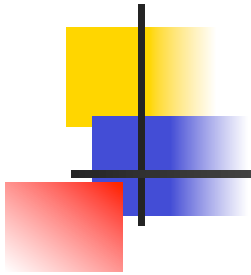
Canvas of a UDP client
C API

```
char Message[] =«Hello World»;
PF
Socket = socket(AF_INET, SOCK_DGRAM, 0);

ReceiverAddr.sin_addr = gethostbyname(ReceiverName);
ReceiverAddr.sin_port = htons(ReceiverPort);

sendto (Socket, Message, strlen(Message), 0,
        &ReceiverAddr, ReceiverAddrLen)
```

Canvas of a UDP server
C API

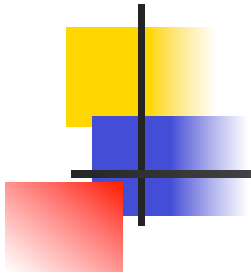
```
#define ReceiverAddrPort 7777
struct sockaddr ReceiverAddr, SenderAddr;
int SenderAddrLen;
char Message[256]
Socket = socket(AF_INET, SOCK_DGRAM,0);
ReceiverAddr.sin_port = htons(ReceiverAddrPort);
bind(Socket, &ReceiverAddr, ReceiverAddrLen);
recvfrom(Socket, Message, sizeof(Message), …,
                &SenderAddr, &SenderAddrLen);
```

TCP and UDP multiplexing



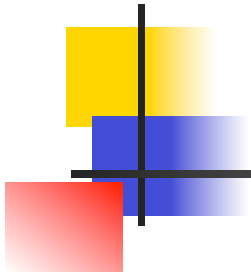
- Having one thread waiting for data per socket may result in a waste of resources
- Accept, read, connect, write blocking operations
- accept is considered as a read operation
- connect is considered as a write operation
- select() function allows to wait on several file descriptors simultaneously for read/write events
- Typically a Leader waits on select() and dispatchs the handling of events to Followers.

TCP and UDP multiplexing with select()



- SetLen= select(SetLenMax, &R_Set, &W_Set, &E_Set, Timeout)
  - SetLen last file to be ready for R/W ops
  - SetLenMax last file to be tested for R/W ops
  - R_Set set specifying files to test for R events
  - W_Set set specifying files to test for W events
  - E_Set set specifying files to test for error events
  - Timeout max duration to wait for completion

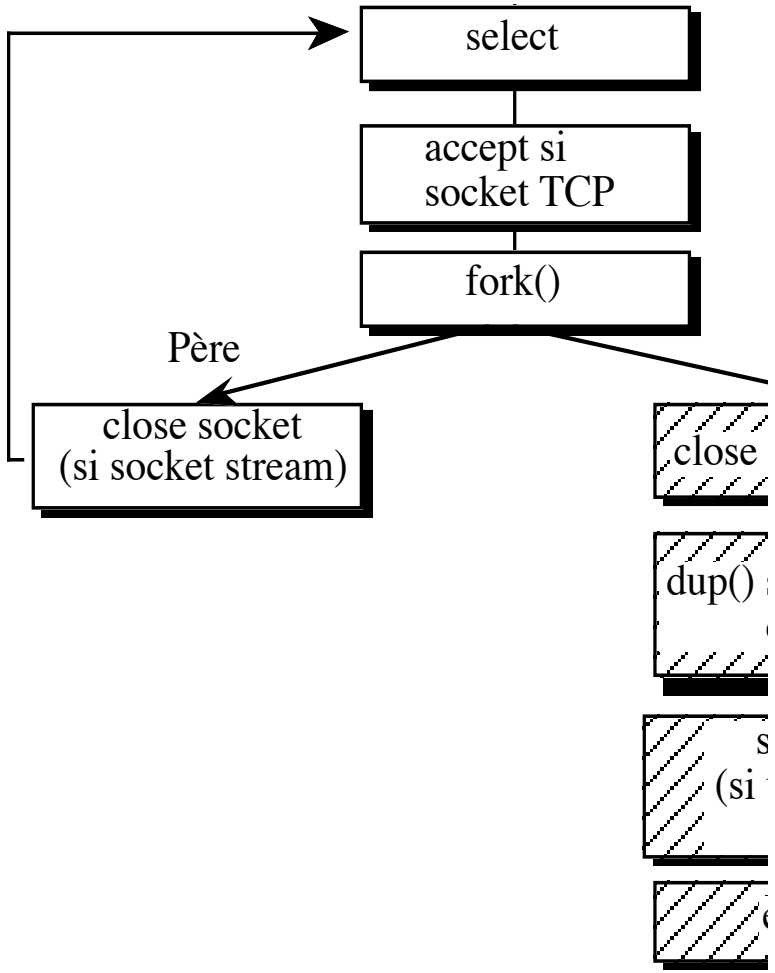
TCP and UDP
bit sets or masks

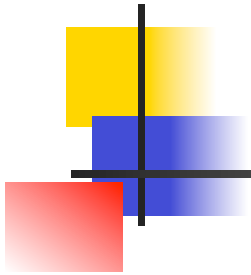


- Bitsets is a simple set data structure
  - n is in the bitset s if (s && 2^n) is true
- FD_CLR(fd, &fdset)
  - Clear the bit for the file fd in the file set fdset.
- FD_ISSET(fd, &fdset)
  - Return a non-zero value if the bit for the file fd is set in the file set pointed to by fdset, and 0 otherwise.
- FD_SET(fd, &fdset)
  - Ses the bit for the file fd in the file set fdset.
- FD_ZERO(&fdset)
  - Initialise the file set fdset to have zero bits for all files.

TCP and UDP super-servers or daemons

- Daemons start at boot time and respond to network requests, hardware activity, or other programs by performing some task.
- inetd (internet service daemon) is a super-server daemon on many Unix systems that provides Internet services.
- Typical use of select()

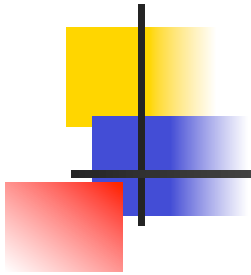




API C : other utilities

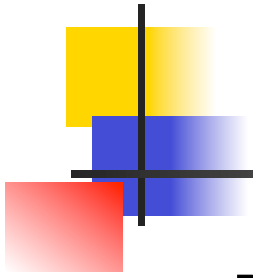
- Retrieve information about hosts (blocking read operations)
  - gethostbyaddr(struct sockaddr *HostAddr, int HostAdddrLen, int Type);
  - gethostbyname(char *HostName),
  - gethostent()
- Retrieve local information about ports (tcp/udp, id, name)
  - getservbyport(int Port, char *Proto)
  - getservbyname(char *Nom, char *Proto)
- Retrieve information about address or port of the specified socket
  - getsockname(Socket, &sa, &len)
  - getpeername(Socket, &sa, &len)
- Shutdown socket (all or part)
  - shutdown(Socket, Direction)
  - close(Socket)

Heterogenous systems endianess



- Endianness designates the byte ordering in memory
- A big-endian system stores the most significant byte of a word at the smallest memory address
- For internet protocols, the network order is big-endianness.
- Functions convert 16-bit and 32-bit integers between network byte order and host byte order
  - htonl(net_long host_long)
  - htons(net_short host_short)
  - ntohl(host_long net_long)
  - ntohs(host_short net_short)

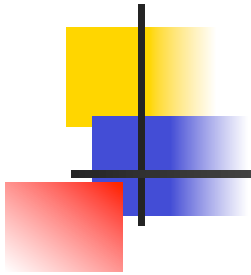
TCP and UDP summary on C API



- Powerfull API:
  - Multicast, asynchronous behaviour (O_NONBLOCK), etc
- … but requires additional tools …
  - Executor services,
- … data conversions …
  - htons,
- … and a lot of programming …

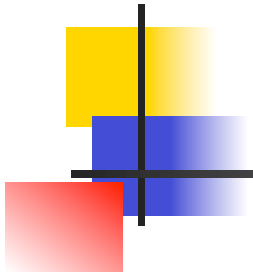
```
sockaddr.sin_family          = AF_INET;
sockaddr.sin_addr.s_addr  = INADDR_ANY;
sockaddr.sin_port            = htons (PORT_SERV);
bind(sock, (struct sockaddr *)&sockaddr, sizeof(sockaddr));
```

Canvas of a TCP server
Java API



- Java API close to C API but easier to use
- ServerSocket: listening socket
- Socket: communication socket

```
// Create socket and bind using constructor
ServerSocket  serverSocket=  new ServerSocket(Port);

// Block until accept a client
Socket socket = serverSocket.accept();
```



Canvas of a TCPserver
Java API in a concurrent system (1/3)

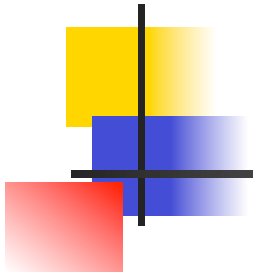
- Use threads to lower complexity of concurrent management
- The server multiplexes communications using threads created on the fly or using a design pattern such as Executor Service

```
ServerSocket serverSocket = new ServerSocket(Port);
Socket socket = serverSocket.accept();
new Thread_Dialogue(socket).start();
```

Canvas of a TCP server
Java API for concurrent systems (2/3)

- Use of stream to « hide » complexity

```
class Pong extends Thread {
    Socket s;
    public Pong(Socket socket){s = socket;}
    public void run(){
        BufferedReader in = new BufferedReader(
            new InputStreamReader(s.getInputStream()))
        PrintWriter out = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(s.getOutputStream(), true);
        …
    }
}
```



Canvas of a TCP server

Java API in a concurrent system (3/3)

```
class Pong extends Thread {
    public void run(){
        PrintWriter out = …
        BufferedReader in = …
        while (true) {
            String str = in.readLine(); // read text on input socket
            System.out.println(str);    // output text on terminal
            out.println(str);           // write text on output socket
        }                               // ping – pong application
    }
}
```

Canvas of a TCP client
Java API

- Client part of the ping pong application

```
class Ping {
  public static void main(String args[]){
    Socket s = new Socket(args[1], port);
    BufferedReader in = new BufferedReader(
       new InputStreamReader(s.getInputStream()));
    PrintWriter out = new PrintWriter(new BufferedWriter(
      new OutputStreamWriter(s.getOutputStream())), true);
    String str =«Hello World!»;
    for (int i = 0; i < 10; i++) {
       out.println(str);            // read text from input socket
       System.out.println (str); // output text on terminal
       str = in.readLine();         // write text on output socket
    }
```

Canvas of a UDP server
Java API

```
DatagramSocket  socket;
DatagramPacket  message = null;
byte[]          buffer;
InetAddress     address;
int             port = 0;
// Initialisations
socket          = new DatagramSocket(PORT_SERV_UDP);
buffer          = new byte[256];
message         = new DatagramPacket(buffer, buffer.length);
// Wait for a message
socket.receive(message);
// Send back message or reply
address         = message.getAddress();
port            = message.getPort();
Message = new DatagramPacket(buffer, buffer.length, address, port);
socket.send(message);
```

Canvas of a UDP client
Java API

```
DatagramSocket  socket;
DatagramPacket  message;
byte[]          buffer;
InetAddress     address;

// Initialisations
buffer          = new byte[256];
socket          = new DatagramSocket();
address         = InetAddress.getByName(…);
// Emission
packet          = new DatagramPacket(buffer, buffer.length, address, port);
socket.send(packet);
// Reception
Packet           = new DatagramPacket(buffer, buffer.length);
socket.receive(packet);
String message = new String(packet.getData());
```