U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Comunicações Industriais

Industrial Communications

2022/2023

Support information for lab assignment 1a

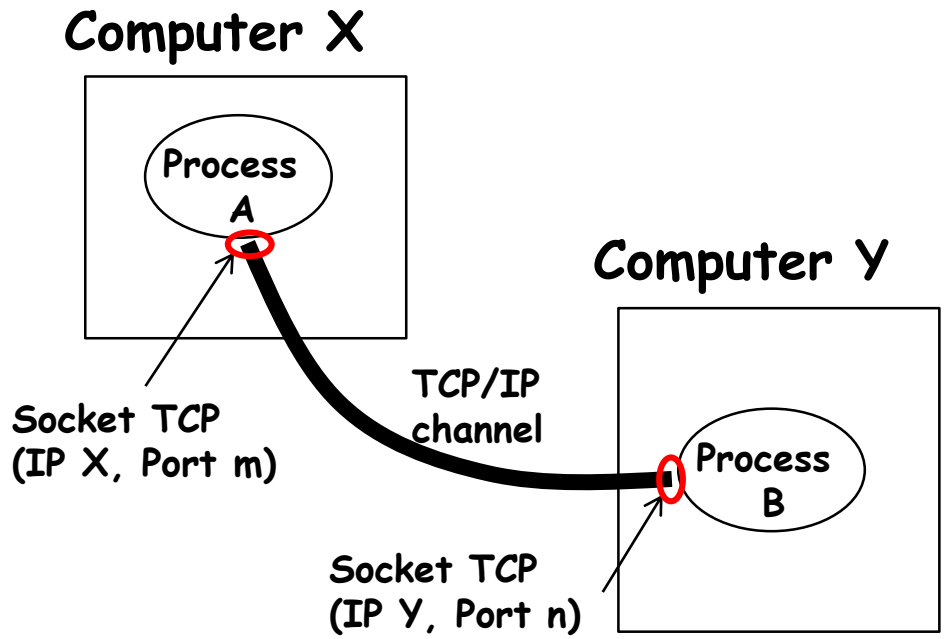
TCP sockets in C

Learning objectives

- Understand the concept of communication channel.
- Understand the construction of channels with *sockets.*
- Ability to create TCP sockets in C for IP networks.

Communication channels and sockets

- **Sockets**
  - technology to create **logical communication channels** between **processes** (normally in different computers)
  - Sockets = channel end-points



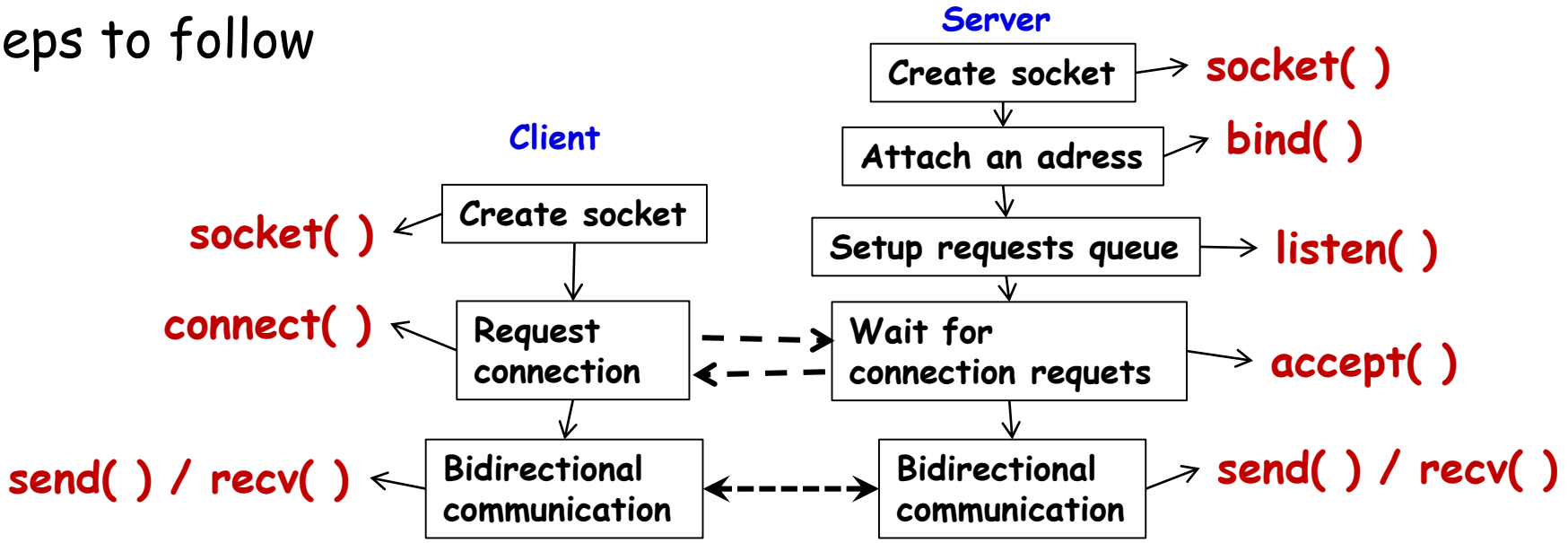
Communication channels and sockets

- **Sockets**
  - Exist for multiple protocol families, protocols and channel types
- We will use just
  - **IP** (Internet) protocol family and the **TCP** protocol
  - Programming language: **C** environment: **Linux** (or Cygwin)
    - » Libraries **<sys/socket.h> <netinet/in.h> <sys/types.h>**
    - » Online manual pages, many resources on the web...

Creating a TCP channel

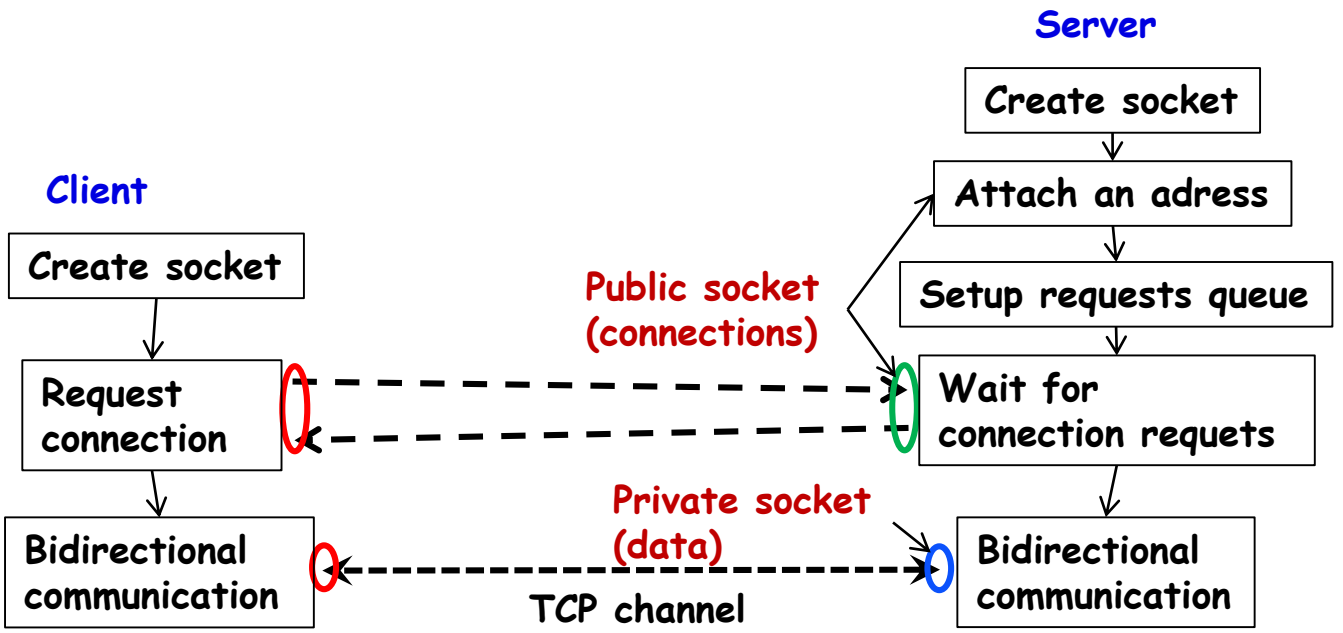
- **TCP channels**
  - Follow a **connection-oriented** model
    - » **First** set up a **connection** and **only after** data can be **communicated**
    - » Connections are **asymmetric**
      Server: passive, waits for connection requests
      Client: active, takes the initiative to request connections

- Steps to follow



Channels and TCP sockets

- **TCP channels**
  - Identified by the (IP Address, TCP Port) pairs of their end-points
    - » **Server:** socket to wait for **connections** + socket for **data communication**
    - » **Client:** uses same socket



Creating a socket

- int **socket** (int domain, int type, int protocol)
  - **domain**: protocol family"- in practice **PF_INET** for the Internet;
  - **type**: channel properties – in practice **SOCK_STREAM** forTCP;
  - **protocol**: which protocol – in practice 0 = **IPPROTO_TCP**

  - returns a **socket local identifier**

  Used in the server and the client

Attaching an address to a socket

- int **bind** (int sockfd, struct sockaddr *my_addr, socklen_t addrlen)
  - **sockfd**: socket local identifier (returned by *socket()*)
  - **my_addr**: pointer to structure with adress to attach
  - **addrlen**: length of the structure pointed to by *my_addr*

  Mandatory in the server (to publicize the server)

  Optional in the client (normally not used)

Defining addresses and data types

- struct **sockaddr_in** { /* socket address */
  sa_family_t **sin_family**; /* address family: AF_INET */
  u_int16_t **sin_port**; /* port in network byte order */
  struct in_addr **sin_addr**; /* internet address */
  };
- struct **in_addr** { /* Internet address */
  u_int32_t **s_addr**; /* 32 bit address in network byte order */
  };
- **Converting formats: network ←→ host**
  - unsigned long int **htonl** (unsigned long int hostlong);
  - unsigned short int **htons** (unsigned short int hostshort);
  - unsigned long int **ntohl** (unsigned long int netlong);
  - unsigned short int **ntohs** (unsigned short int netshort);
- **Converting addresses: network ←→ dotted decimal**
  - long **inet_aton**(char *, struct in_addr *)
  - char ***inet_ntoa**(struct in addr)

Example of defining a socket address

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
...
#define MYPORT 22222
...
int s;
struct sockaddr_in sad_loc;
...

sad_loc.sin_family = AF_INET;

sad_loc.sin_port = htons(MYPORT);

inet_aton("127.0.0.1", &sad_loc.sin_addr) ;
```

Wait for, and accept, connection requests

- int **listen** (int s, int backlog)
  - **s**: local identifier of the socket that receives the connection requests
  - **backlog**: connection requests queue capacity
  - returns immediatly with 0 →OK or -1 → error
- int **accept** (int s, struct sockaddr *addr, socklen_t *addrlen)
  - **s**: local identifier of the socket that waits for the connection requests
  - **addr**: pointer to a *struct sockaddr* with the address of the remote socket
  - **addrlen**: pointer to an integer with the size of *addr
  - Returns an **identifier** of the data communication socket

Used in the server side, only

Requesting connections

- int **connect** (int sock, struct sockaddr *s_addr, socklen_t addrlen)
  - **sock**: local identifier of the client socket
  - **s_addr**: pointer to a *struct sockaddr* with address of the server socket
  - **addrlen**: integer with size of *s_addr
  - Blocks until the connection is set up (accepted by the server)

Used in the client side,only

Sending and receiving data

- int **send** (int s, const void *buf, size_t len, int flags)
- int **recv** (int s, const void *buf, size_t len, int flags)

  - **s**: identifier of the socket for data communication
    buf: pointer to a buffer
    with the data to transmit / where to place the received data
  - **len**: length in bytes of the data to transmit / maximum data to receive
  - **flags**: integer (bit array) specifying several options – just use 0

  - returns number of bytes efectively sent /receive or -1 if error

  Used in both sides, client and server

Closing a connection

- int **close** (int s)
  - **s**: local identifier of the socket
  - The socket is destroyed as soon as there is no data in transmission

- int **shutdown** (int s, int how)
  - **s**: local identifier of the socket
  - **how**: 0 - inhibits reception, 1 - inhibits transmission, 2 - inhibits both

  Used in both sides, client and server

A few details

- bind, accept, connect – require a socket address of type ***sockaddr***
- IPv4 addresses are better specified with type ***sockaddr_in***
  - which can be cast to *sockaddr*
- Thus (example with *connect*):

```
struct  sockaddr_in  serv;
socklen_t   addrlen = sizeof(serv);
...
connect (sock, (struct sockaddr *) &serv, addrlen )
```

A few details

- send, recv – (TCP) operate on a **stream** model
  - there is no concept of "message".
- Thus, the parts of data that we get with **recv** can be different than those we sent with **send**
  - Ex. **send**("abcdefgh") and in the first **recv** receive "abcd", only, needing another **recv** to get the remainder "efgh".
  - However, this might be irrelevant – depends on the application
- accept – allows knowing who is connecting
  - through the socket address that is passed in the parameters
  - but frequently it is not needed

Example of a sever

```
#include ...
#define ......

main() {
        int so, sd, len;
        struct sockaddr_in  loc, rem;
        socklen_t  addlen = sizeof(loc);
        char buf[BUF_LEN];

        so = socket(PF_INET, SOCK_STREAM, 0);

        loc.sin_family = AF_INET;
        loc.sin_port = htons(MYPORT);
        inet_aton("127.0.0.1",&loc.sin_addr);

        bind(so, (struct sockaddr *) &loc, addlen);

        listen(so, 10);
        sd=accept(so, (struct sockaddr *) &rem, &addlen);

        len=recv(sd, buf, BUF_LEN, 0);
        ...
}
```

Revisit slides 5 & 6

Example of a client

```
#include ...
#define ......

main () {
        int sock, len;
        struct sockaddr_in  serv;
        socklen_t  addlen = sizeof(serv);
        char buf[BUF_LEN];

        sock = socket(PF_INET, SOCK_STREAM, 0);

        serv.sin_family = AF_INET;
        serv.sin_port = htons(MYPORT);
        inet_aton("127.0.0.1",&serv.sin_addr);

        connect(sock, (struct sockaddr *) &serv, addlen);

        scanf("%s", buf);
        len=send(sock, buf, strlen(buf)+1, 0);
        ...
}
```

Revisit slides 5 & 6