

# Redes de Computadores LEIC-A

## Socket Programming

Prof. Paulo Lobato Correia

IST, DEEC – Área Científica de Telecomunicações

1

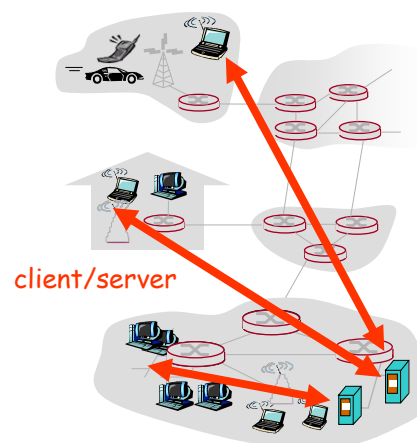
## Client-Server Architecture

### Server:

- Always-on host;

### Clients:

- Initiate communication with server, specifying server's **IP address** and **port number**;
- May be intermittently connected;
- Do not communicate directly with each other.

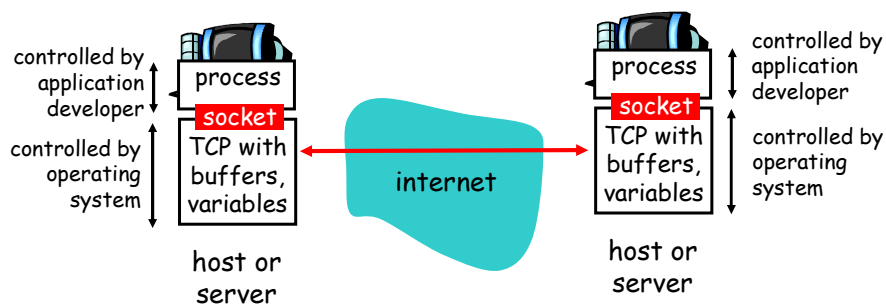


2

## Socket Programming using TCP

Socket: a **door** between application process and the end-end-transport protocol (UDP or TCP);

TCP service: reliable transfer of **bytes** from one process to another.



3

## Client/server Socket Interaction: TCP

### Client

```
create socket,
connect to hostid, port=x
clientSocket =
  socket()
connect ()

send request using
clientSocket

read reply from
clientSocket

close
clientSocket
```

### Server (running on **hostid**)

```
create socket,
port=x, for
incoming request:
welcomeSocket =
  socket()

wait for incoming
connection request
connectionSocket =
  accept()

read request from
connectionSocket

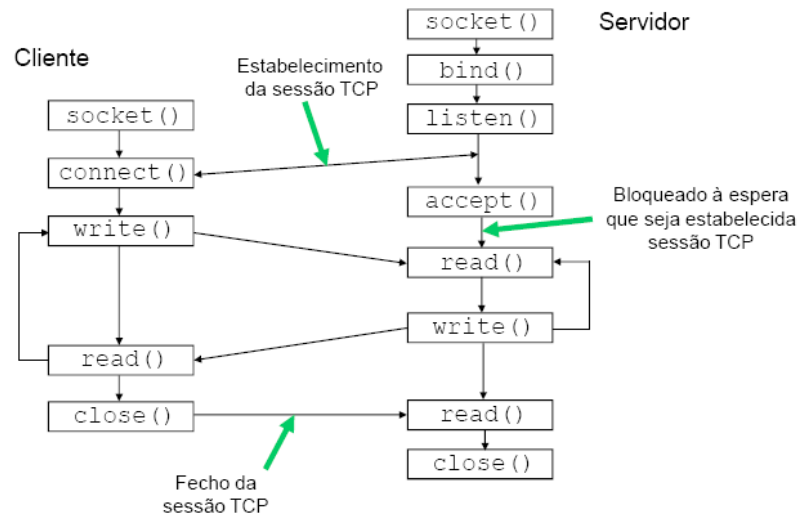
write reply to
connectionSocket

close
connectionSocket
```

TCP  
connection setup

4

## Client/server Socket Interaction: TCP



RC – Prof. Paulo Lobato Correia 5

5

**TCP Client**

```

#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#define PORT "58001"

...
int fd, errcode;
ssize_t n;
socklen_t addrlen;
struct addrinfo hints, *res;
struct sockaddr_in addr;
char buffer[128];
...

fd=socket(AF_INET, SOCK_STREAM, 0); //TCP socket
if (fd==-1) exit(1); //error

memset(&hints, 0, sizeof hints);
hints.ai_family=AF_INET; //IPv4
hints.ai_socktype=SOCK_STREAM; //TCP socket

errcode=getaddrinfo("tejo.technico.ulisboa.pt", PORT, &hints, &res);
if (n!=0) /*error*/ exit(1);

n=connect(fd, res->ai_addr, res->ai_addrlen);
if (n==-1) /*error*/ exit(1);

n=write(fd, "Hello!\n", 7);
if (n==-1) /*error*/ exit(1);

n=read(fd, buffer, 128);
if (n==-1) /*error*/ exit(1);

write(1, "echo: ", 6); write(1, buffer, n); //output to screen
...
freeaddrinfo(res);
close(fd);

```

**TCP Server**

```

...
fd=socket(AF_INET, SOCK_STREAM, 0); //TCP socket
if (fd==-1) exit(1); //error

memset(&hints, 0, sizeof hints);
hints.ai_family=AF_INET; //IPv4
hints.ai_socktype=SOCK_STREAM; //TCP socket
hints.ai_flags=AI_PASSIVE;

errcode=getaddrinfo(NULL, PORT, &hints, &res);
if (errcode != 0) /*error*/ exit(1);

n=bind(fd, res->ai_addr, res->ai_addrlen);
if (n==-1) /*error*/ exit(1);

if (listen(fd, 5)==-1) /*error*/ exit(1);
...
while(1) {
    addrlen=sizeof(addr);
    if ((newfd=accept(fd, (struct sockaddr*)&addr, &addrlen))!=-1)
        /*error*/ exit(1);

    n=read(newfd, buffer, 128);
    if (n==-1) /*error*/ exit(1);
    write(1, "received: ", 10); write(1, buffer, n);

    n=write(newfd, buffer, n);
    if (n==-1) /*error*/ exit(1);

    close(newfd);
}
...
freeaddrinfo(res);
close(fd);

```

*blocks until connection from client* (points to `accept()`)

*connection establishment TCP three-way handshake* (points to `connect()` in client code)

6

## Socket Programming with UDP

UDP – no “connection” between client and server:

- No handshaking;
- Sender explicitly includes IP address and port of destination to each packet;
- Server must extract IP address and port of client from the received packet.

UDP – transmitted data may be received out of order, or lost!

*UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server*

## Client/server Socket Interaction: UDP

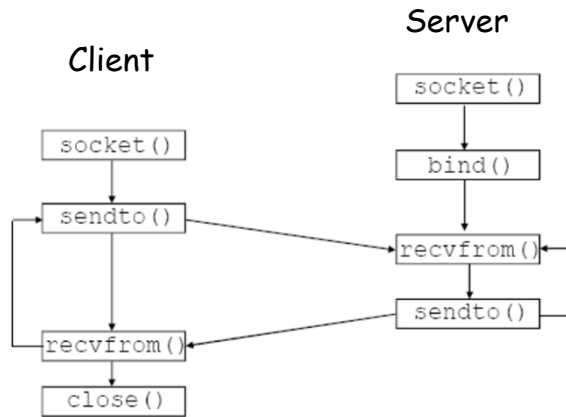
### Client

```
create socket,
clientSocket =
socket(...,SOCK_DGRAM,...)
↓
Create datagram with server IP and
port=x; send datagram via
clientSocket
↓
read datagram from
clientSocket
↓
close
clientSocket
```

### Server (running on hostid)

```
create socket,
port= x.
serverSocket =
socket(...,SOCK_DGRAM,...)
↓
read datagram from
serverSocket
↓
write reply to
serverSocket
specifying
client address,
port number
```

## Client/server Socket Interaction: UDP



RC – Prof. Paulo Lobato Correia 9

9

**TÉCNICO LISBOA**

### UDP Client

```

#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#define PORT "58001"
...
int fd, errcode;
ssize_t n;
socklen_t addrlen;
struct addrinfo hints, *res;
struct sockaddr_in addr;
char buffer[128];
...

fd=socket(AF_INET, SOCK_DGRAM, 0); //UDP socket
if(fd==-1) /*error*/ exit(1);

memset(&hints, 0, sizeof hints);
hints.ai_family=AF_INET; //IPv4
hints.ai_socktype=SOCK_DGRAM; //UDP socket

errcode=getaddrinfo("tejo.tecnico.ulisboa.pt", PORT, &hints, &res);
if(errcode!=0) /*error*/ exit(1);

n=sendto(fd, "Hello!\n", 7, 0, res->ai_addr, res->ai_addrlen);
if(n==-1) /*error*/ exit(1);
...
addrlen=sizeof(addr);
n=recvfrom(fd, buffer, 128, 0, (struct sockaddr*)&addr, &addrlen);
if(n==-1) /*error*/ exit(1);

write(1, "echo: ", 6); write(1, buffer, n);
...
freeaddrinfo(res);
close(fd);

```

### UDP Server

```

...
fd=socket(AF_INET, SOCK_DGRAM, 0); //UDP socket
if(fd==-1) /*error*/ exit(1);

memset(&hints, 0, sizeof hints);
hints.ai_family=AF_INET; // IPv4
hints.ai_socktype=SOCK_DGRAM; // UDP socket
hints.ai_flags=AI_PASSIVE;

errcode=getaddrinfo(NULL, PORT, &hints, &res);
if(errcode!=0) /*error*/ exit(1);

n=bind(fd, res->ai_addr, res->ai_addrlen);
if(n==-1) /*error*/ exit(1);

while (1){
    addrlen=sizeof(addr);
    n=recvfrom(fd, buffer, 128, 0, (struct sockaddr*)&addr, &addrlen);
    if(n==-1) /*error*/ exit(1);
    write(1, "received: ", 10); write(1, buffer, n);
    ...
    n=sendto(fd, buffer, n, 0, (struct sockaddr*)&addr, addrlen);
    if(n==-1) /*error*/ exit(1);
}
...
freeaddrinfo(res);
close(fd);

```

blocks until datagram received from a client

10

## Socket Programming: TCP vs UDP

### TCP:

- `read()` and `write()`;
- Byte stream (and no byte is lost);
- Bytes read with `read()` may correspond to several `write()`;
- Bytes written with `write()` may need to be read with several `read()`;

### UDP:

- `sendto()` and `recvfrom()`;
- Preserves boundary between messages;
- Each message read with `recvfrom()` corresponds to a single `sendto()`;
- A message may be lost.

### Login

Username: alunos  
Password: alunos

### Compilação

`gcc prog.c -o prog`

makefile

```
all: rc
rc: prog.c
    gcc prog.c -o prog
```

`make`

### Manuais on-line

exemplo: `man 2 read`  
`man 3 printf`

`tcpecho + udpecho:`  
`tejo.tecnico.ulisboa.pt, porto 58001`