Universidade de Coimbra
Faculdade de Ciências e Tecnologia

DEPARTAMENTO DE ENG. ELETROTÉCNICA E COMPUTADORES
Redes de Computadores

Class Assignment nº2
Sockets TCP and UDP Sockets
2020/2021

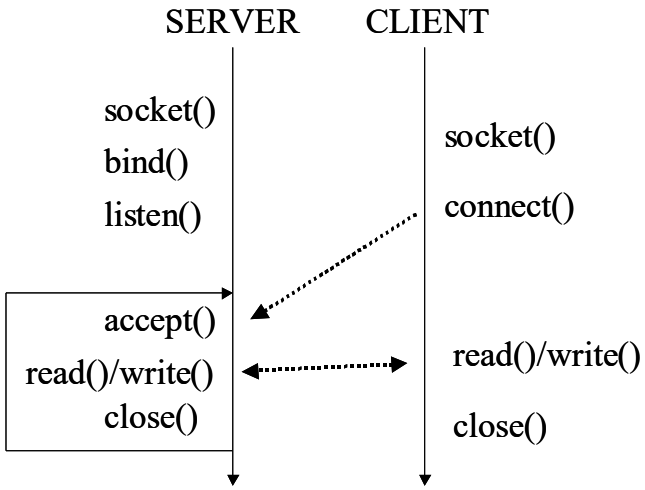
---

1 - Introduction

In this lesson we will learn how to use TCP sockets and UDP sockets.
Sockets are important structures in the TCP/IP architecture. Sockets allow communication between two different processes on the same or different devices connected to the Internet.
TCP is a connection-oriented protocol, whereas UDP is a connectionless protocol. A key difference between TCP and UDP is speed, as TCP is comparatively slower than UDP. Overall, UDP is a much faster, simpler, and efficient protocol, however, retransmission of lost data packets is only possible with TCP.

2 - Programming using TCP sockets

TCP sockets are used for communication between a server and a client process. The server's code runs first, which opens a port and listens for incoming connection requests from clients. Once a client connects to the same (server) port, the client or server may send a message. Once the message is sent, whoever receives it (server or client) will process it accordingly.



TCP Code

```
#include <sys/types.h>
#include <sys/socket.h>

/* Cria um novo socket. Página de manual no Linux: "man socket" */
int socket(int domain, int type, int protocol)

domain:     Domínio no qual o socket será usado
            (processos Unix / internet)
            (AF_UNIX, AF_INET, AF_INET6, ...)
type:       Tipo de ligação (orientada a ligações ou datagrama)
            (SOCK_STREAM, SOCK_DGRAM, ...)
protocol:   Forma de comunicação (0 para protocolo default)

DEVOLVE: "Descritor de socket"
```

```
#include <sys/types.h>
#include <sys/socket.h>

/* Associa um socket a um determinado endereço. Página de manual no
Linux: "man 2 bind" */
int bind(int fd, const struct sockaddr *address, socklen_t
address_len)

fd:         "Descritor de socket"
address:    Ponteiro para o endereço a associar ao socket
address_len:      Dimensão da estrutura de dados indicada em <address>

DEVOLVE: 0 para sucesso, -1 para erro

Internet Sockets:

struct sockaddr_in {
      short             sin_family;       // AF_INET
      u_short           sin_port;         // porto a associar
      struct in_addr    sin_addr;         /* INADDR_ANY=qualquer
endereço do host */
      char              sin_zero;         // padding, deixar em branco
}

Nota:
1) O domínio Internet usa a estrutura struct sockaddr_in em vez da
estrutura struct sockaddr. De acordo com o POSIX as funções devem
fazer um cast (conversão do tipo de dados) das struct sockaddr_in para
struct sockaddr, de modo a poderem usar as funções de sockets.
2) "INAAR_ANY – quando especificado na função bind, o socket ficará
ligado a todas as interfaces locais da máquina
```

```
#include <sys/types.h>
#include <sys/socket.h>

/* Aguardar pela recepção de ligações. Página de manual no Linux: "man
listen" */
int listen(int fd, int backlog)

fd:         "Descritor de socket"
backlog:    Quantos clientes são mantidos em espera (a aguardar o
            accept) antes de haver recusa de ligação (com a mensagem
            "Connection Refused")

DEVOLVE: 0 para sucesso, -1 para erro
```

```
#include <sys/types.h>
#include <sys/socket.h>

/* Aceita uma ligação. Página de manual no Linux: "man 2 accept" */
int accept(int fd, const struct sockaddr *address,
      socklent_t* address_len)

fd:         "Descritor de socket"
address:    Estrutura de dados que vai ser preenchida com informação
            sobre a ligação que está a ser estabelecida
address len:      Comprimento do buffer <address>. No final da chamada
            irá conter o tamanho (em octetos) da estrutura <address>

DEVOLVE:    "Descritor de socket" da ligação aceite, -1 em caso de erro
```

```
#include <sys/types.h>
#include <sys/socket.h>

/* Inicia uma ligação num socket. Página de manual no Linux: "man
connect" */
int connect(int fd, const struct sockaddr *address,
      socklen_t address_len)

fd:         "Descritor de socket"
address:    Endereço do servidor ao qual se pretende ligar
address len:      Dimensão da estrutura <address>

DEVOLVE:     0 para ligação estabelecida, -1 no caso contrário
```

```
#include <sys/types.h>
#include <sys/socket.h>

/* converte nome para endereço. Página de manual no Linux: "man
gethostbyname" */
struct hostent * gethostbyname(const char* name)

DEVOLVE:    Estrutura com o endereço Internet correspondente ao nome

/* Outras funções relevantes. Consultar a página de manual respetiva
no Linux com o comando "man <nome_função>"
Nota: Na arquitetura i80x86 a ordem dos bytes é a little-endian
(primeiro é armazenado na memória o byte menos significativo),
enquanto que as comunicações em rede utilizam (enviam) primeiro os
bytes mais significativos (big-endian). */

#include <arpa/inet.h> ou <netinet/in.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);

htonl() -converte um inteiro sem sinal da ordem de bytes do host para
a ordem de bytes da rede.

htons() -converte um inteiro short sem sinal da ordem de bytes do host
para a ordem de bytes da rede.

ntohl() -converte um inteiro sem sinal da ordem de bytes da rede para
a ordem de bytes do host.

ntohs() -converte um inteiro short sem sinal netshort da ordem de
bytes da rede para a ordem de bytes do host.
```

```
#include <arpa/inet.h> ou <netinet/in.h>

in_addr_t inet_addr(const char *cp);

Converte um endereço IPv4 na notação xxx.xxx.xxx.xxx em binário,
usando a ordem de bytes usada na rede. Este valor pode ser armazenado
directamente no campo sin_addr.s_addr da struct sockaddr_in.
```

TCP Example

```
/*****************************************************************
 * SERVIDOR no porto 9000, à escuta de novos clientes.  Quando surjem
 * novos clientes os dados por eles enviados são lidos e descarregados no ecran.
 *****************************************************************/
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <netdb.h>
#include <string.h>

#define SERVER_PORT 9000
#define BUF_SIZE      1024

void process_client(int fd);
void erro(char *msg);

int main() {
  int fd, client;
  struct sockaddr_in addr, client_addr;
  int client_addr_size;

  bzero((void *) &addr, sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = htonl(INADDR_ANY);
  addr.sin_port = htons(SERVER_PORT);

  if ( (fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
       erro("na funcao socket");
  if ( bind(fd,(struct sockaddr*)&addr,sizeof(addr)) < 0)
       erro("na funcao bind");
  if( listen(fd, 5) < 0)
       erro("na funcao listen");

  while (1) {
    client_addr_size = sizeof(client_addr);
    client = accept(fd,(struct sockaddr *)&client_addr, (socklen_t *)&client_addr_size);
    if (client > 0) {
      if (fork() == 0) {
        close(fd);
        process_client(client);
        exit(0);
      }
    close(client);
    }
  }
  return 0;
}

void process_client(int client_fd)
{
       int nread = 0;
       char buffer[BUF_SIZE];

       nread = read(client_fd, buffer, BUF_SIZE-1);
       buffer[nread] = '\0';
       printf("%s\n", buffer);
       fflush(stdout);

       close(client_fd);
}

void erro(char *msg)
{
       printf("Erro: %s\n", msg);
       exit(-1);
}
```

```
/***********************************************************
 * CLIENTE liga ao servidor (definido em argv[1]) no porto especificado
 * (em argv[2]), escrevendo a palavra predefinida (em argv[3]).
 * USO: >cliente <enderecoServidor>  <porto>  <Palavra>
 ***********************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>

void erro(char *msg);

int main(int argc, char *argv[]) {
  char endServer[100];
  int fd;
  struct sockaddr_in addr;
  struct hostent *hostPtr;

  if (argc != 4) {
       printf("cliente <host> <port> <string>\n");
       exit(-1);
  }

  strcpy(endServer, argv[1]);
  if ((hostPtr = gethostbyname(endServer)) == 0)
       erro("Nao consegui obter endereço");

  bzero((void *) &addr, sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = ((struct in_addr *)(hostPtr->h_addr))->s_addr;
  addr.sin_port = htons((short) atoi(argv[2]));

  if((fd = socket(AF_INET,SOCK_STREAM,0)) == -1)
       erro("socket");
  if( connect(fd,(struct sockaddr *)&addr,sizeof (addr)) < 0)
       erro("Connect");
  write(fd, argv[3], 1 + strlen(argv[3]));
  close(fd);
  exit(0);
}

void erro(char *msg)
{
       printf("Erro: %s\n", msg);
       exit(-1);
}
```

Exercises using TCP sockets:

Exercise 1:

Modify the previous client and server applications so that the client sends a text sentence to the server, and the latter should return the same sentence to the client but inverted.

Exercise 2:

Modify the previous client and server applications, so that the server presents the following options to each client that connects to it:

1 – Capital of Portugal
2 – Capital of Spain
3 – Capital of France

The client must select option 1, 2 or 3, and the server must return the respective answer to the client application.

3 - Programming using UDP sockets

In the previous exercise, we used of IP communications based on the TCP Protocol (Transmission Control Protocol) at the transport layer level. In this exercise we will explore the use of communications with the UDP protocol (User Datagram Protocol).

In UDP there are no connections and therefore it is not necessary to maintain state information regarding associations between computers. This means that a UDP server does not accept connections and therefore a UDP client does not need to establish a connection to the server. UDP packets are sent independently, without any guarantees regarding their delivery or ordering upon arrival at the destination device.

When creating the UDP socket, the type (socket_type) must indicate the use of datagrams instead of data streams:

```
#include <sys/types.h>
#include <sys/socket.h>

/* Cria um novo socket */
int socket(int domain, int type, int protocol)

domain:     Domínio no qual o socket será usado
(processos Unix / internet)
(AF_UNIX ou AF_INET)

type:  Tipo de ligação (orientada a ligações ou utilizando datagramas)
(SOCK_STREAM ou SOCK_DGRAM)

protocol:     Forma de comunicação (0)

DEVOLVE: Descritor de socket
```

In addition of creating the socket itself, it is necessary to use the bind function on the server to define the port to be used, as well as other auxiliary functions and structures already described in the previous exercise. The server can receive UDP packets from multiple clients.

A program will use the *sendto* and *recvfrom* functions (among others) to send or receive UDP packets from another computer. These functions receive or return the address and the port of the other device:

```
#include <sys/types.h>
#include <sys/socket.h>
```

/* Recebe uma mensagem através de um *socket* */
int recvfrom(int sockfd, void *buf, int len, int flags,
struct sockaddr *src_addr, socklen_t *addrlen)

sockfd: *socket* onde é recebida a mensagem.
buf: *buffer* para armazenamento da mensagem.
len: número máximo de *bytes* a ler (de acordo com o tamanho do *buffer*).
flags: controlo da operação de leitura (utilizar comando "man recvfrom" no linux para mais informação).
src_addr: estrutura para armazenamento do endereço de origem da mensagem.
addrlen: tamanho do endereço de origem da mensagem.

A função devolve: em caso de sucesso o número de bytes (tamanho da mensagem UDP) recebidos.

```
#include <sys/types.h>
#include <sys/socket.h>

/* Envia uma mensagem */
int sendto(int sockfd, void *buf, int len, int flags,
        struct sockaddr *dest_addr, socklen_t addrlen)
```

Parâmetros: semelhantes aos da função anterior (ver página de manual com o comando "man sendto" no Linux).
A função devolve: em caso de sucesso o número de bytes enviados.

Server UDP example:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

#define BUFLEN 512   // Tamanho do buffer
#define PORT 9876    // Porto para recepção das mensagens

void erro(char *s)
```

```
{
        perror(s);
        exit(1);
}

int main(void)
{
        struct sockaddr_in si_minha, si_outra;

        int s, slen = sizeof(si_outra) , recv_len;
        char buf[BUFLEN];

        // Cria um *socket* para recepção de pacotes UDP
        if((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
        {
                erro("Erro na criação do socket");
        }

        si_minha.sin_family = AF_INET;
        si_minha.sin_port = htons(PORT);
        si_minha.sin_addr.s_addr = htonl(INADDR_ANY);

        // Associa o *socket* à informação de endereço
        if(bind(s,(struct sockaddr*)&si_minha, sizeof(si_minha)) == -1)
        {
                erro("Erro no bind");
        }

        // Espera recepção de mensagem (a chamada é bloqueante)
        if((recv_len = recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &si_outra, &slen)) == -
1)
        {
                erro("Erro no recvfrom");
        }
        // Para ignorar o restante conteúdo (anterior do buffer)
        buf[recv_len]='\0';

        printf("Recebi uma mensagem do sistema com o endereço %s e o porto %d\n",
                inet_ntoa(si_outra.sin_addr), ntohs(si_outra.sin_port));
        printf("Conteúdo da mensagem: %s\n" , buf);

        // Fecha *socket* e termina programa
        close(s);
        return 0;
}
```

Exercise 1:

Now you should use the application of the previous example and the program “netcat” as a client for receiving and sending UDP messages, respectively.

Usage syntax (on Linux):
nc <-u> <Endereço IP servidor> <Porto>

Exercise 2:

Now you should develop a set of client-server programs for accessing and consulting a people's phone numbers.
To do so, you must create a main program that will launch two distinct processes: a UDP server process that will be responsible for receiving new records (“name – phone number” pairs) and registering them in a text file; and a TCP server process that, after validating each user by login and password, must return the respective telephone number for each name sent by the TCP client program.
Every time any TCP client program connects to the TCP server, the latter must send the message:

> “ Welcome to the phone records program!
> Login: "

After the user enters his login, the server should ask him for the password. If the login and password match, the TCP server should receive the person's name, open the records text file, search for the name and send the associated telephone number to the client.