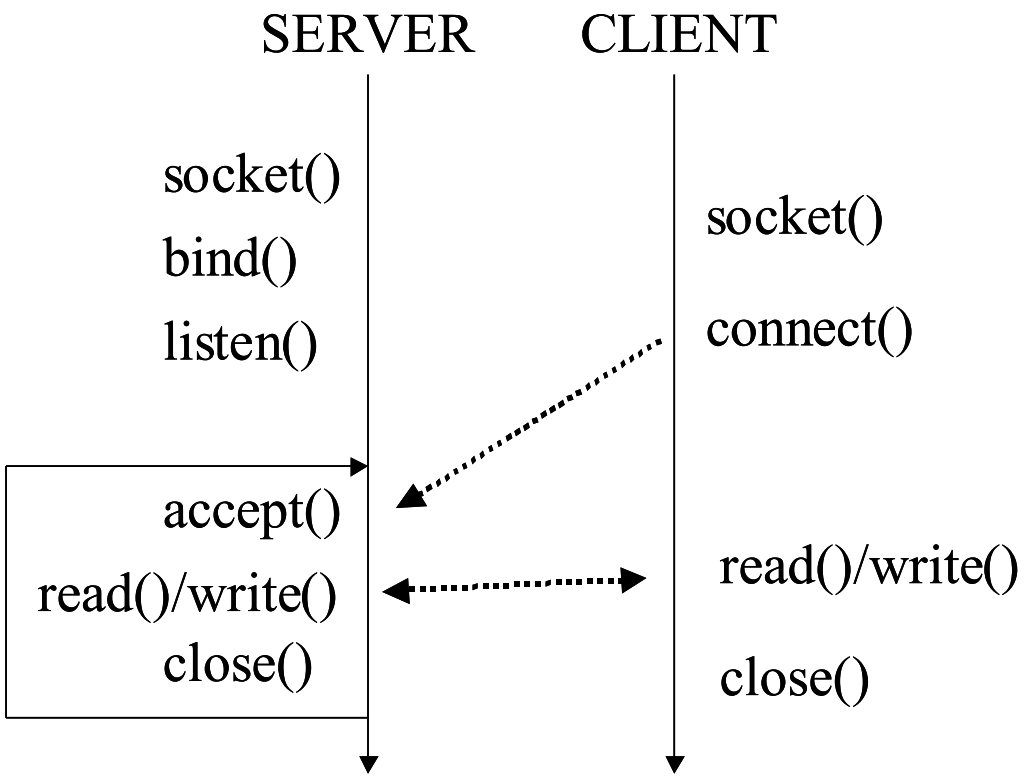
Ficha 4 - Sockets

Ano Lectivo de 2015/2016

Programação com Sockets (TCP)

- Surgiu no BSD Unix 4.1c, 1980.
- Modelo de programação de rede utilizado em virtualmente todos os sistemas operativos (Windows, Unix, ...)
- Existem dois tipos principais:
  - Unix Sockets (FIFOs/Pipes no sistema de ficheiros)
  - Internet
- Modelo:



```
#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*)
- protocolo *default* para SOCK_STREAM = TCP
- protocolo *default* para SOCK_DGRAM = UDP

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: 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.

```
#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*.

```
/*****************************************************************
 * 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);
}
```

Exercícios de Programação com *sockets* (TCP):

Exercício 1:

Modifique as aplicações cliente e servidor apresentadas, de modo a que o servidor devolva ao cliente (por rede) a mensagem de texto dele recebida mas sem vogais.

Programação com *sockets* (UDP)

Ao contrário do TCP (analisado anteriormente), no UDP não existem ligações e consequentemente não é necessário manter informação de estado relativamente a associações entre computadores. Isto significa que um servidor UDP não aceita ligações e da mesma forma um cliente UDP não tem a necessidade de estabelecer uma ligação ao servidor. Os pacotes UDP são enviados isoladamente entre sistemas, sem quaisquer garantias em relação à sua entrega ou ordenação na chegada ao sistema de destino.

- Na criação do *socket* o tipo (*socket_type*) deve indicar a utilização de *datagrams* em vez de *data streams*:

```
#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 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 do socket criado
```

- Para além da criação do *socket* propriamente dito, é necessário utilizar a função *bind* no servidor para definir a porta a utilizar, bem como outras

funções e estruturas auxiliares já descritas anteriormente. Um servidor pode receber pacotes UDP de vários clientes.

- Um programa pode utilizar as funções *sendto* e *recvfrom* (entre outras) para enviar ou receber pacotes UDP de outro computador. Estas funções recebem ou devolvem o endereço e porto do outro computador:

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

/* Recebe uma mensagem através de um socket. Página de manual no Linux:
"man recvfrom" */
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, em caso de falha -1.

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

/* Envia uma mensagem. Página de manual no Linux: "man sendto" */
int sendto(int sockfd, void *buf, int len, int flags,
        struct sockaddr *dest_addr, socklen_t addrlen)
```

Parâmetros: semelhantes aos da função anterior.

A função devolve: em caso de sucesso o número de bytes enviados, em caso de falha -1.

Exemplo de programa para um servidor que recebe mensagens UDP:

```
#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;
}
```

Exercícios de Programação com *sockets* (UDP):

Exercício 2:

Utilize a aplicação do exemplo anterior e o programa "netcat" como cliente para enviar uma mensagen UDP.

Síntaxe de utilização (no Linux):
nc <-v> <-u> <Endereço IP servidor> <Porto>

Exercício 3:

Utilizando a aplicação do exemplo anterior construa um programa que funcione como cliente para o envio de uma mensagem UDP ao servidor. A síntaxe de utilização do programa cliente deverá ser a seguinte:

cliente <endereço IP servidor> <porto> <mensagem>

Exemplo de utilização:
cliente 127.0.0.1 9876 "Hello world!"

Nota: Pode seguir a estrutura do programa do exemplo anterior e as funções *socket* e *sendto* anteriormente discutidas.

Exercício 4 (opcional):

Utilizando o *wireshark* (http://www.wireshark.org/) observe as mensagens UDP enviadas do cliente para o servidor, bem como o seu conteúdo e restante informação.

Caso a aplicação *wireshark* não esteja instalada fazer: "`sudo apt-get install wireshark`". Para correr fazer "`sudo wireshark`", de modo a ter acesso a todas as portas.