TROCA DE MENSAGENS
SOCKETS

- Comunicando processos através de SOCKETS
- SOCKETS com conexão
- SOCKETS sem conexão

SOCKETS

Princípios dos sockets:

2. Fornecer uma interface geral permitindo a construção de aplicações baseadas em rede,
3. Suportar comunicação entre processos não relacionados residindo localmente ou remotamente,
4. Permitir a várias formas de comunicação:
   - Comunicação internet (TCP, UDP)
   - Comunicação na mesma máquina
   - Comunicação *Broadcast* e *Multicast*

SOCKETS: Introdução

Quando criamos um SOCKET o *domínio de comunicação* é especificado. Os domínios especificam o escopo da comunicação, local ou remota, e como os nomes e endereços são formados e interpretados nas chamadas subsequentes, caminhos ou endereços IP/numero de porta..

UNIX: sockets tem nome de arquivos e são usados com processos do mesmo *host*. Os processos cliente e servidor comunicam-se via arquivos (PIPE). (AF_UNIX)

Internet: sockets permitem que processos não relacionados de diferentes *hosts* comuniquem-se entre si, usando endereços como: “128.195.1.1@port 21”. (AF_INET)

SOCKETS: Introdução

Os processos devem concordar com um conjunto de regras de comunicação que são determinadas por *protocolos*.

Algumas famílias (agrupamento) de protocolos:

TCP - Transmission Control Protocol IP - Internet Protocol

UDP - User Datagram Protocol

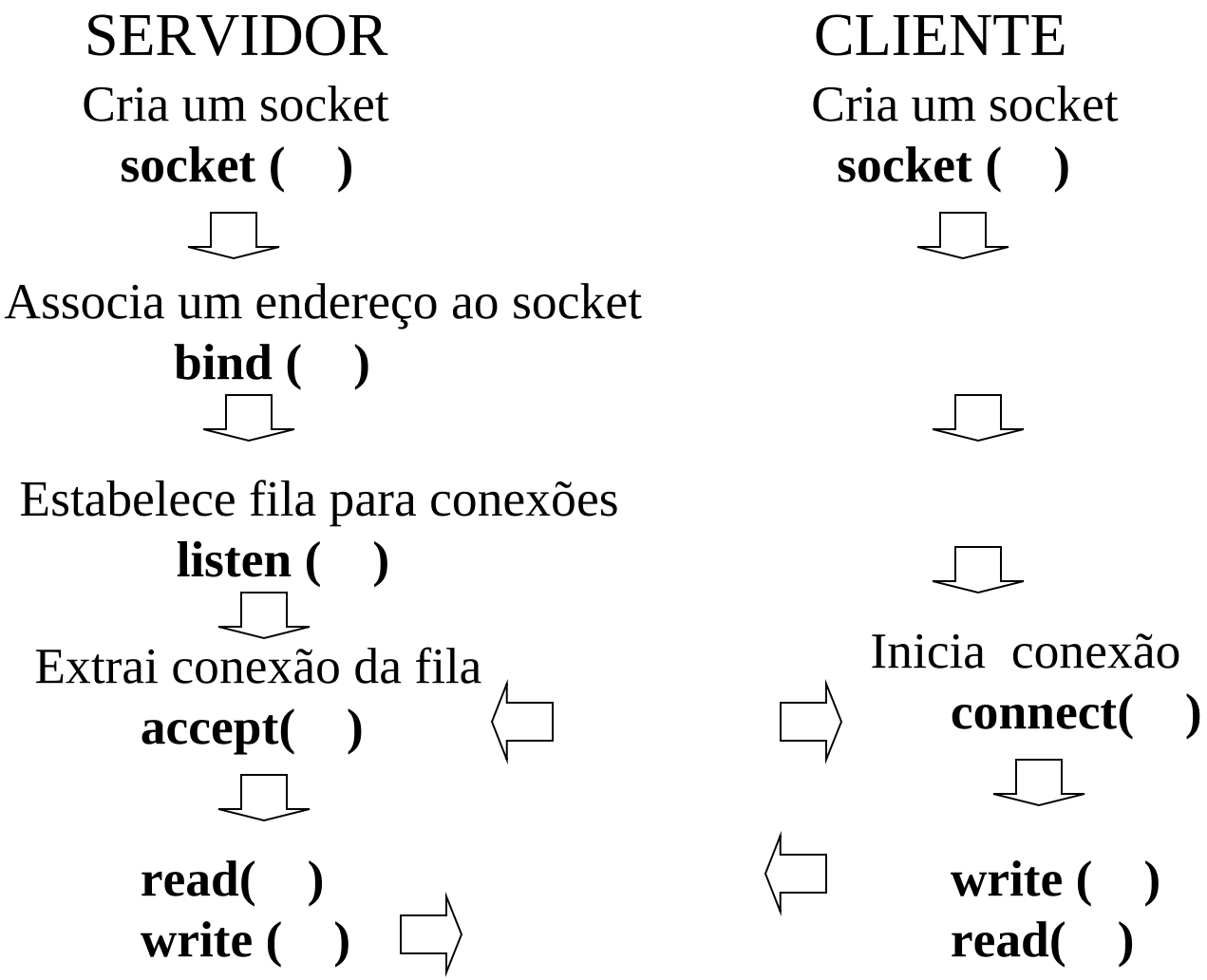
SOCKETS: Introdução

Tipos de sockets + comuns : *stream* e *datagram*

Stream: confiáveis, dados são entregues em ordem na mesma sequencia de envio. Comunicação bidirecional, orientado a conexão. Existe uma conexão lógica entre os processos. Informações sobre a conexão estabelecida antes da transmissão. Mensagens urgentes.

Datagram: não confiáveis, dados podem ser recebidos fora de ordem. Comunicação bidirecional, porém sem conexão. Cada datagrama é enviado separadamente podendo usar caminhos diferentes. Sem controle de fluxo. Controle de erro mínimo. Pacotes pequenos e de tamanho fixo.

Sockets - Orientado a Conexão



Especificação de endereço para Sockets

- **struct sockaddr** : estrutura genérica para todas as famílias de protocolos

```
struct sockaddr {
        u_short  sa_family;
        char     sa_data[14];
        };
```

struct sockaddr_un : endereço para família de protocolos domínio UN

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

struct   sockaddr_un {
        short    sun_family;              /*AF_UNIX*/
                 char     sun_path[108];
                 };

sin_addr;
```

Especificação de endereço Internet

- **struct in_addr** : estrutura com um endereço Internet (4 bytes)

```
struct in_addr {
        uint32_t s_addr;  /* endereço IP, formato rede*/
};
```

- **struct sockaddr_in** : endereço para a família de protocolos internet

```
struct sockaddr_in {
        short sin_family; /*AF_INET*/
        u_short sin_port;       /*porta formato rede*/
        struct in_addr sin_addr; /* endereço IP*/
        char  sin_zero[8];      /*não usado*/
};
```

Socket

<table>
<tr><td>Include
<sys/types.h>
<sys/socket.h></td><td colspan="3"></td></tr>
<tr><td>Chamada</td><td colspan="3">int socket(int familia,
int tipo,
int protocolo);</td></tr>
<tr><td rowspan="2">retorno</td><td>sucesso</td><td>falha</td><td>errno</td></tr>
<tr><td>0 e 1 desc.</td><td>-1</td><td>sim</td></tr>
</table>

connect

<table>
<tr><td>Include
<sys/types.h>
<sys/socket.h></td><td colspan="3"></td></tr>
<tr><td>Chamada</td><td colspan="3">int connect(int socket,
struct sockaddr *nome,
int *tamnome);</td></tr>
<tr><td rowspan="2">retorno</td><td>sucesso</td><td>falha</td><td>errno</td></tr>
<tr><td>0</td><td>-1</td><td>sim</td></tr>
</table>

bind

Include <sys/types.h> <sys/socket.h>			
Chamada	int bind(int socket, const struct sockaddr *nome, int tamnome);		
retorno	sucesso	falha	*errno*
	0	-1	sim

listen

Include <sys/types.h> <sys/socket.h>			
Chamada	int listen(int socket, int tam_fila);		
retorno	sucesso	falha	*errno*
	0	-1	sim

accept

Include <sys/types.h> <sys/socket.h>			
Chamada	int accept(int socket, struct sockaddr *end, int *tamend);		
retorno	sucesso	falha	*errno*
	inteiro ns	-1	sim

read() / write()

Include <unistd.h>			
Chamada	ssize_t read(int fd, const void *buffer, size_t count); ssize_t write(int fd, const void *buffer, size_t count);		
retorno	sucesso	falha	*errno*
	bytes	-1	sim

Exemplo Domínio UNIX (socklcli.c socklser.c)

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#define NAME "my_socket"

main(void)
{
        extern int errno;

        int sd,                 /* descritor do socket */
            ns,i;
        static char buf[256];  /* buffer de mensagens */
        static struct sockaddr_un serv_sock; /* endereço UNIX
para serv */
        int fromlen;

        if (( sd = socket( AF_UNIX, SOCK_STREAM,0 )) < 0) {
                printf("erro na geração do socket");
                exit(1);
        }
```

```
        serv_sock.sun_family = AF_UNIX;    /* comunicação UNIX */
        strcpy(serv_sock.sun_path, NAME); /* nome do socket */

        /* conecta nome do socket */
        if ( connect( sd, (struct sockaddr *) &serv_sock,
                sizeof(serv_sock.sun_family)
+strlen(serv_sock.sun_path)) < 0)  {
           printf("erro de conexao");
           exit(1);
        }
        for (i=1; i<=5; i++) {
           sprintf(buf, "Cliente enviando mensagem n.: %d\n",i);
           write( sd, buf, sizeof(buf));
        }
        printf("mensagens enviadas = %d\n",i);
        close(sd);
        exit(0);
}
```

```
#define NAME "my_socket"

main(void)
{
        extern int errno;
        int sd,                 /* descritor do socket */
            ns,ns1,                     /* novo descritor */
            er,i;
        static char buf[256];
        static struct sockaddr_un sock_cli,
                                  sock_cli1,
                                  sock_ser;
        int fromlen, fromlen1;

        void clean_up (int, char *);

        if ((sd = socket( AF_UNIX, SOCK_STREAM,0 )) < 0) {
           printf("erro de geração do socket \n");
           exit(1);
        }
        sock_ser.sun_family = AF_UNIX;
        strcpy( sock_ser.sun_path, NAME );
        unlink( NAME );
```

```
        /*  bind nome */

         if ( bind( sd, (struct sockaddr *) &sock_ser,
           sizeof(sock_ser.sun_family)
+strlen(sock_ser.sun_path)) < 0) {
        printf("erro de bind \n");
        clean_up( sd,NAME );
        exit(2);
        }

        listen( sd,2 );
        fromlen = sizeof( sock_cli );

        if ((ns = accept( sd, (struct sockaddr *) &sock_cli,
                &fromlen)) < 0) {
        printf("erro na conexão \n");
        clean_up(sd, NAME );
        exit(3);
        }
        if ((ns1 = accept( sd, (struct sockaddr *) &sock_cli1,
                  &fromlen1)) < 0) {
        printf("erro na conexao 1 \n");
        clean_up(sd, NAME);
        exit(3);
        }
```

```
for (i=1; i<=10; i++) {
          sleep(1);
          if (( er = read(ns, buf, sizeof(buf))) <= 0 )
               printf("sem mensagem \n");
          else printf("s-> %s", buf);
          sleep(1);
          if ((er = read(ns1, buf, sizeof(buf))) <= 0 )
               printf("sem mensagem \n");
          else printf("s-> %s", buf);
        }
        close(ns);
        close(ns1);
        clean_up( sd, NAME);
        exit(0);
}
void clean_up( int sc, char *file ) {
        close (sc);
        unlink( file );
}
```

Exemplos :

- Domínio Internet (sockrcli.c sockrser.c)

Neste domínio os processos devem ter informação de endereço e porta para comunicar-se. Uma aplicação pode conhecer o nome do *host* (venus) mas não tem informações específicas (endereço internet, serviços oferecidos(portas), etc.). Para isto existem chamadas ao serviço de rede.

Getpeername / getsockname

Include <sys/socket.h>	<sys/types.h>		

Chamada	int getpeername(int sockfd, struct sockaddr *peer, socklen_t *addrlen);
	
	int getsockname(int sockfd, struct sockaddr *local, socklen_t *addrlen);		

retorno	sucesso	falha	*errno*
	endereço em peer ou local	NULL	sim

inet_pton / inet_ntop

<table>
<tr><td>Include</td><td colspan="3"><sys/types.h>
<sys/socket.h><netdb.h>
<netinet/in.h></td></tr>
<tr><td>Chamada</td><td colspan="3">int inet_pton(int family, const char *stptrr, void *addrptr);

char *inet_ntop(int family, const void *addrptr, char *stptrr, size_t len);</td></tr>
<tr><td rowspan="2">retorno</td><td>sucesso</td><td>falha</td><td>errno</td></tr>
<tr><td>1
ponteiro</td><td>0, -1
NULL</td><td>sim</td></tr>
</table>

Gethostbyname (gethost.c)

<table>
<tr><td>Include</td><td colspan="3"><sys/types.h>
<netdb.h> <sys/socket.h>
<netinet/in.h></td></tr>
<tr><td>Chamada</td><td colspan="3">struct hostent
*gethostbyname(const char *nome);</td></tr>
<tr><td rowspan="2">retorno</td><td>sucesso</td><td>falha</td><td>errno</td></tr>
<tr><td>referência
a hostent</td><td>NULL</td><td>sim</td></tr>
</table>

```
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

main(void)
{
    struct hostent *host;
    static char who[10];

        printf("Informe o nome do host: ");
        scanf("%10s",who);
        host = gethostbyname( who );
        if ( host != (struct hostent *) NULL ) {
          printf("Informações sobre %s :\n",who);
          printf("Nome : %s\n", host->h_name);
          printf("Aliases : ");
          while ( *host->h_aliases ) {
                printf("%s ", *host->h_aliases );
                ++host->h_aliases;
                }
```

```
        printf("\n Tipo de endereço : %i\n", host->h_addrtype);
        printf("Tamanho do endereço : %i\n", host->h_length);
        printf("Lista de endereços : ");
        while ( *host->h_addr_list ) {
              struct in_addr in;
              memcpy( &in.s_addr, *host->h_addr_list,
                                        sizeof(in.s_addr));
              printf("[%s] = %s ", *host->h_addr_list,
                    inet_ntoa(in));
              ++host->h_addr_list;
        }
        printf("\n");
      }
}
```

Getservbyname (getserv.c)

<table>
<tr><td>Include
<netdb.h></td><td colspan="3"><sys/types.h>
<sys/socket.h>
<netinet/in.h></td></tr>
<tr><td>Chamada</td><td colspan="3">struct servent
*getservbyname(const char *nome,
char *protocolo);</td></tr>
<tr><td rowspan="2">retorno</td><td>sucesso</td><td>falha</td><td>errno</td></tr>
<tr><td>referência
a servent</td><td>NULL</td><td></td></tr>
</table>

```
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

main(void)
{
    struct servent *serv;
    static char protocol[10], service[10];

        printf("Informe o nome do serviço: ");
        scanf("%9s",service);
        printf("Informe protocolo: ");
        scanf("%9s",protocol);
        serv = getservbyname ( service, protocol );
```

```
        if ( serv != (struct servent *) NULL ) {
          printf("Informações que encontrei  :\n");
          printf("Nome : %s\n", serv->s_name);
          printf("Aliases : ");
          while ( *serv->s_aliases ) {
                printf("%s ", *serv->s_aliases );
                ++serv->s_aliases;
                }
          printf("\n Numero da Porta : %i\n",htons( serv-
>s_port ));
          printf("Familia de protocolo : %i\n", serv->s_proto);
        } else
          printf("Serviço %s para protocolo %s não
                                encontrado\n",service,protocol);
}
```

```
/*
        servidor exemplo internet
*/
#include "local.h"


main(void)
{
        extern int errno;
        int sd,                 /* descritor do socket */
            ns,                 /* novo descritor */
            erro,len,i;
        static char buf[256];
         struct sockaddr_in sock_cli, /* enderecos */
                            sock_ser;
        int fromlen;            /* tamanho end. cliente */

        if ((sd = socket( AF_INET, SOCK_STREAM,0 )) < 0) {
           printf("erro de geração do socket \n");
           exit(1);
        }
        memset( &sock_ser, 0, sizeof(sock_ser) );
        sock_ser.sin_family    = AF_INET;              /* tipo */
        sock_ser.sin_addr.s_addr= htonl(INADDR_ANY); /* interface */
        sock_ser.sin_port      = htons(PORT);          /* fake port */
```

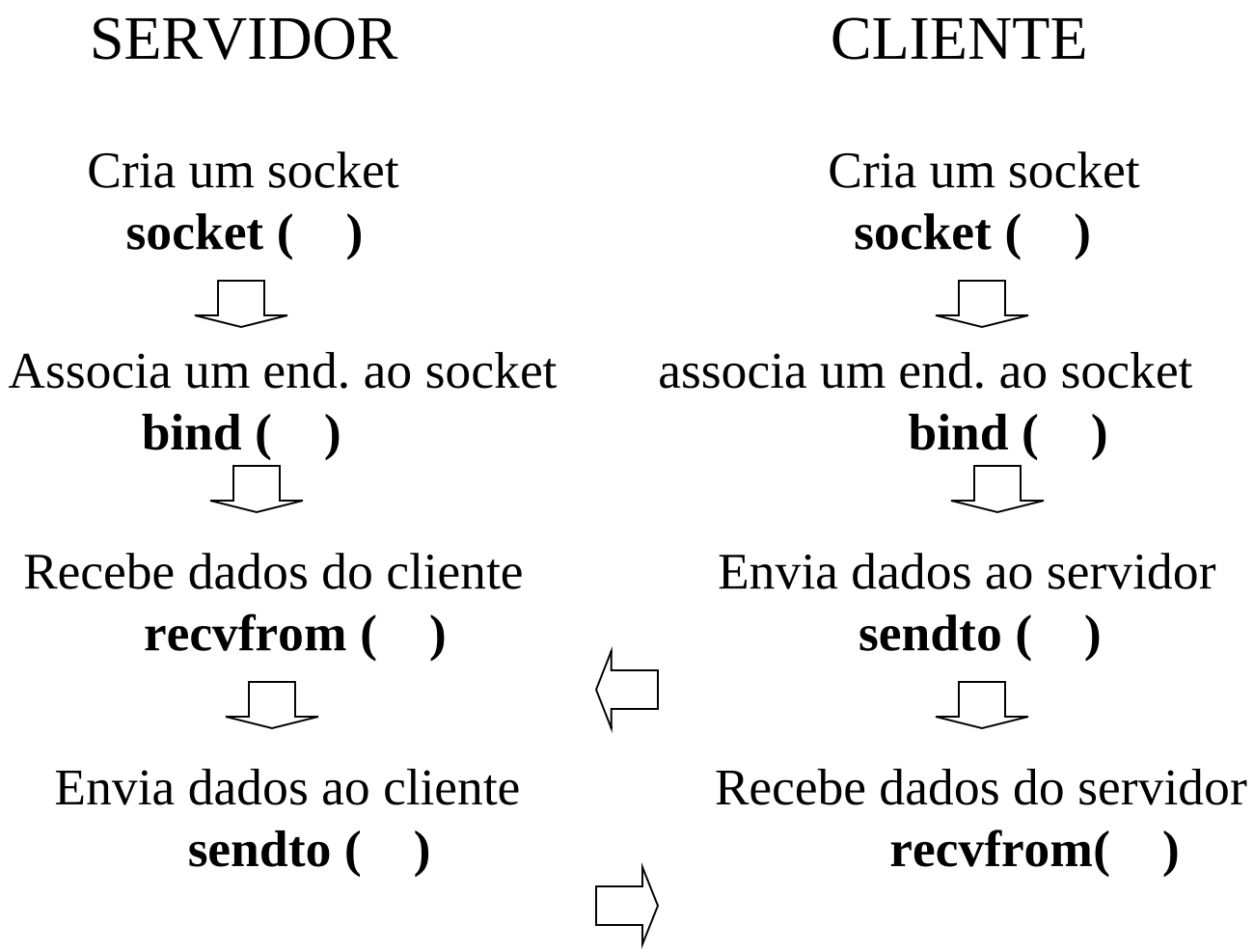
```
  /*  bind nome */
 if ( bind( sd, (struct sockaddr *) &sock_ser,
    sizeof(sock_ser)) < 0) {
  printf("erro de bind \n");
  close(sd);
  exit(2);
  }
     if ( listen( sd,5 ) < 0 ) {
     printf("erro no listen \n");
     exit(3);
  }
  do {
     fromlen = sizeof(sock_cli);
     if ((ns = accept( sd,(struct sockaddr *) &sock_cli,
           &fromlen)) < 0) {
        printf("erro na conexão \n");
        close(sd);
        exit(3);
     }
     if ( fork() == 0 ) {                    /* processo filho */
          while ( (len=read(ns, buf, BUFSIZ)) > 0 ) {
            for (i=0; i < len; ++i)
              buf[i] = toupper(buf[i]);
            write(ns, buf, len);
            if ( buf[0] == '.') break;
          }
          close(ns);
          exit(0);
       } else close(ns);                     /* processo pai */
     } while( 1 );
}
```

```
/*
  cliente implementaçao internet
*/
#include "local.h"

main( int argc, char *argv[] )
{
        extern int errno;

        int sd,                 /* descritor do socket */
            ns,i,len;
        static char buf[256];  /* buffer de mensagens */
        struct sockaddr_in
                        serv_sock; /* endereço UNIX para serv */
        struct hostent *host;          /* servidor */

        if ( argc != 2 ){
           fprintf(stderr, "uso: %s servidor\n",argv[0]);
           exit(1);
        }
        host = gethostbyname(argv[1]);        /* informacoes servidor */
        if (host == (struct hostent *) NULL ) {
                printf("erro no gethostbyname \n");
                exit(2);
        }
        memset(&serv_sock, 0, sizeof(serv_sock));
        serv_sock.sin_family = AF_INET;              /* tipo */
```

```
    memcpy(&serv_sock.sin_addr, host->h_addr,host->h_length);
    serv_sock.sin_port = htons(PORT);    /* fake port */

    if (( sd = socket( AF_INET, SOCK_STREAM,0 )) < 0) {
            printf("erro na geração do socket");
            exit(3);
    }

    /* conecta nome do socket */
    if ( connect( sd, (struct sockaddr *) &serv_sock,
                  sizeof(serv_sock)) < 0)  {
       printf("erro de conexao");
       exit(4);
    }
    do {
        write(fileno(stdout),"> ", 3);    /* prompt */
        if (( len=read(fileno(stdin), buf, BUFSIZ)) > 0) {
            write(sd, buf, len);
            if ((len=read(sd, buf, len)) > 0 )
              write(fileno(stdout), buf, len);
        }
       }while (buf[0] != '.');
    close(sd);
    exit(0);
}
```

```
/*  arquivo local.h   */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define PORT 6996
static char buf[BUFSIZ];
```

Sockets - Sem Conexão



send , sendto e sendmsg

<table>
<tr><td>Include
<sys/uio.h></td><td colspan="3"><sys/types.h>
<sys/socket.h></td></tr>
<tr><td>Chamada</td><td colspan="3">int send(int socket, const char *msg,
int tam, int flags);
int sendto(int socket, const char *msg,
int tam, int flags,
const struct sockaddr *to, int tam);
int sendmsg(int socket,
const struct msghdr *msg, int flags);</td></tr>
<tr><td rowspan="2">retorno</td><td>sucesso</td><td>falha</td><td>errno</td></tr>
<tr><td>bytes</td><td>-1</td><td>sim</td></tr>
</table>

recv , recvfrom e recvmsg

Include <sys/uio.h>	<sys/types.h> <sys/socket.h>		
Chamada	int recv(int socket, const char *buffer, int tam, int flags); int recvfrom(int socket, const char *buffer, int tam, int flags, const struct sockaddr *from, int tam); int recvmsg(int socket, const struct msghdr *msg, int flags);		
retorno	sucesso	falha	*errno*
	bytes	-1	sim

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#define SERVER_FILE "server_socket"

main(void)
{
        extern int errno;
        int sd,                 /* descritor do socket */
            ns,ns1,                     /* novo descritor */
            er,i;
        static char buf[40];
        static struct sockaddr_un sock_cli,sock_ser;
        int fromlen;
        void clean_up (int, char *);

        if ((sd = socket( AF_UNIX, SOCK_DGRAM,0 )) < 0) {
           printf("erro de geração do socket \n");
           exit(1);
        }
        sock_ser.sun_family = AF_UNIX;
        strcpy( sock_ser.sun_path, SERVER_FILE );
        unlink( SERVER_FILE );
```

```
        /*  bind nome */

         if ( bind( sd, (struct sockaddr *) &sock_ser,
           sizeof(sock_ser.sun_family)+strlen(sock_ser.sun_path)) < 0) {
        printf("erro de bind \n");
        clean_up( sd,SERVER_FILE );
        exit(2);
        }

        for (i=1; i<=10; i++) {
          recvfrom(sd, buf, sizeof(buf), 0,
                (struct sockaddr *) &sock_cli, &fromlen);
          printf("s-> %s ",buf);
         }
        clean_up( sd, SERVER_FILE);
        exit(0);
}
void clean_up( int sc, char *file ) {
        close (sc);
        unlink( file );
}
```

```
#define SERVER_FILE "server_socket"

main(void)
{
        extern int errno;

        int sd,                 /* descritor do socket */
            ns,i;
        static char buf[40],    /* buffer de mensagens */
                    client_file[15];
        static struct sockaddr_un
                                serv_sock, /* endereço UNIX para serv */
                                clit_sock;
        int fromlen;
        void clean_up( int, char * );

        serv_sock.sun_family = AF_UNIX;       /* comunicação UNIX */
        strcpy(serv_sock.sun_path, SERVER_FILE); /* nome do socket */

        if (( sd = socket( AF_UNIX, SOCK_DGRAM,0 )) < 0) {
                printf("erro na geração do socket");
                exit(1);
        }
```

```
    sprintf(client_file,"%07d_socket",getpid());
    clit_sock.sun_family = AF_UNIX;       /* comunicação UNIX */
    strcpy(clit_sock.sun_path, client_file); /* nome do socket */

    /* bind socket */
    if ( bind( sd, (struct sockaddr *) &clit_sock,
        sizeof(clit_sock.sun_family)+strlen(clit_sock.sun_path)) < 0){
       printf("erro de bind");
       exit(2);
    }
    for (i=1; i<=10; i++) {
       sleep(1);
       sprintf(buf, "Cliente enviando mensagem n.: %d\n",i);
       sendto( sd, buf, sizeof(buf), 0,
            (struct sockaddr *) &serv_sock, sizeof(struct sockaddr));
    }
    printf("mensagens enviadas = %d\n",i);
    close(sd);
    exit(0);
}
```

```
/*   servidor exemplo internet/datagrama  */
#include "local.h"

main(void)
{
        extern int errno;
        int sd,                 /* descritor do socket */
            ns,                 /* novo descritor */
            erro,i;
         struct sockaddr_in sock_cli, /* enderecos */
                            sock_ser;
        int server_len, client_len;          /* tamanho end. cliente */

        if ((sd = socket( AF_INET, SOCK_DGRAM,0 )) < 0) {
           printf("SERVIDOR: erro de geração do socket \n");
           exit(1);
        }
        sock_ser.sin_family    = AF_INET;             /* tipo */
        sock_ser.sin_addr.s_addr= htonl(INADDR_ANY); /* interface */
        sock_ser.sin_port      = htons(0);            /* fake port */

        /*  bind nome */
         if ( bind( sd, (struct sockaddr *) &sock_ser,
           sizeof(sock_ser)) < 0) {
        printf("SERVIDOR: erro de bind \n");
        exit(2);
        }
```

```
        if ( getsockname(sd, (struct sockaddr *) & sock_ser,
                        &server_len ) < 0) {
           printf("SERVIDOR: erro no getsocketname \n");
           exit(3);
        }
        printf("Servidor usando porta %d\n",ntohs(sock_ser.sin_port));
        while(1){
           client_len = sizeof(sock_cli);
           memset(buf,0,BUFSIZ);
           if ((i=recvfrom(sd, buf, BUFSIZ,0,
                (struct sockaddr *) &sock_cli, &client_len)) < 0) {
                printf("SERVIDOR: erro de recvfrom \n");
                close(sd);
                exit(4);
           }
           write(fileno(stdout),buf,i);
           memset(buf,0,BUFSIZ);
           write(fileno(stdout),"> ",3);
           if (fgets(buf,BUFSIZ,stdin) != NULL) {
                if ((sendto(sd,buf,strlen(buf),0,
                  (struct sockaddr *) &sock_cli, client_len)) < 0) {
                  printf("SERVIDOR: erro sendto\n");
                  close(sd);
                  exit(5);
                }
           }
        }
}
```