Comunicação entre Processos

1. Pipes
2. Fifos
3. Sockets

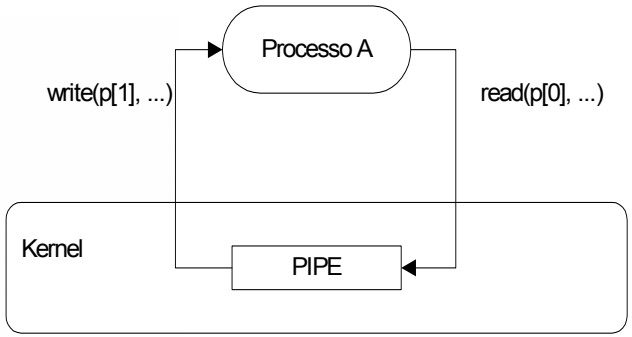
Pipes

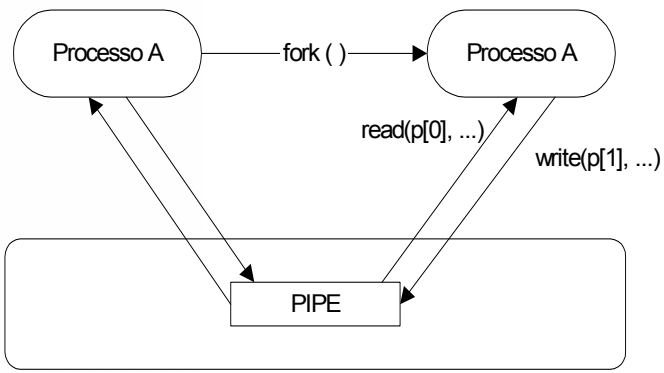
- Implementa um canal de comunicação associado a um processo
- Limitações:
  - Os dados apenas fluem num sentido
  - Só podem ser usados entre processos com um mesmo antepassado

```
#include<unistd.h>

Int pipe(int filedes[2]);     /*devolve  0  se OK
                                         -1 se erro  */
```

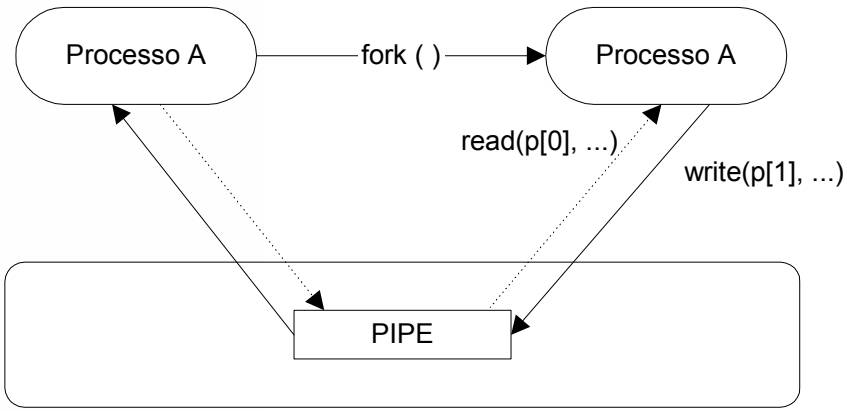
Pipes





- Chamada ao sistema *pipe* cria um par de descritores de ficheiros que referenciam um inode pipe
- Objectivo: comunicação entre processos
- Filho herda todos os dados declarados no processo pai
- Tanto pai como filho conhecem o mesmo pipe
- Se um escrever e o outro ler, a comunicação está estabelecida
- Tanto pai como filho podem ler e escrever do pipe. Cabe ao programador definir o sentido da comunicação.

Pipes



- Comunicação bidireccional precisa de dois pipes
- Leitura de uma mensagem de pipe pipe vazio bloqueia o processo até que lá seja colocada uma mensagem

Quando um dos lados do pipe é fechado:

- Se lermos de um pipe cujo extremo de escrita tenha sido fechado e todos os dados consumidos ➔ read devolve 0 indicando fim de ficheiro.

- Se escrevermos num pipe cujo extremo tenha sido fechado é gerado um sinal SIGPIPE. Se este for ignorado ou a função de tratamento o ignorar, a função write devolve erro sendo o valor igual a EPIPE.

Pipes

Exemplo - Troca de mensagens entre processo pai e filho

```
#define MSGSIZE 25
Char *msg1="Msg entre processos – 1";
Char *msg2="Msg entre processos – 2";
Char *msg3="Msg entre processos – 3";
int main( ){ mensagem
int pid, p[2], i;
Char buff[MSGSIZE];

if pipe(p)<0 {
   perror("Erro na criação do pipe");
   exit(1);
}
if ((Pid= fork ( ))==0) {;
                              /* Processo Filho */
    close(p[0]);
    write(p[1],msg1,MSGSIZE);
    write(p[1],msg2,MSGSIZE);
    write(p[1],msg3,MSGSIZE);
} else
                              /* Processo Pai */
    if (pid > 0) {
                                   /* Processo Pai */
        close(p[1]);
        for( i=0 ; i<3 ; i++ ) {
            read(p[0],buff,MSGSIZE);
            printf("Recebi:%s\n",buff );
        }
    }
    else
            printf(" Erro no fork!...\n");
}
```

Fifos

- Permite a comunicação entre processos sem qualquer relação de parentesco
- Comunicação realizada por um canal de comunicação permanente e acessível a qualquer processo, suportado por um ficheiro especial ➔ FIFO
- Depois de criado (funções *mknode* ou *mkfifo*) abre-se com a função *open*.
- Função de manipulação: *close, read, write, unlink, etc* (funções normais de I/O)

Fifos

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

Int mknod (const char *pathname, mode_t mode, dev_t dev);
                /* 0 se OK, -1 se erro*/
Int mkfifo(const char *pathname, mode_t mode)
                /* 0 se OK, -1 se erro*/
```

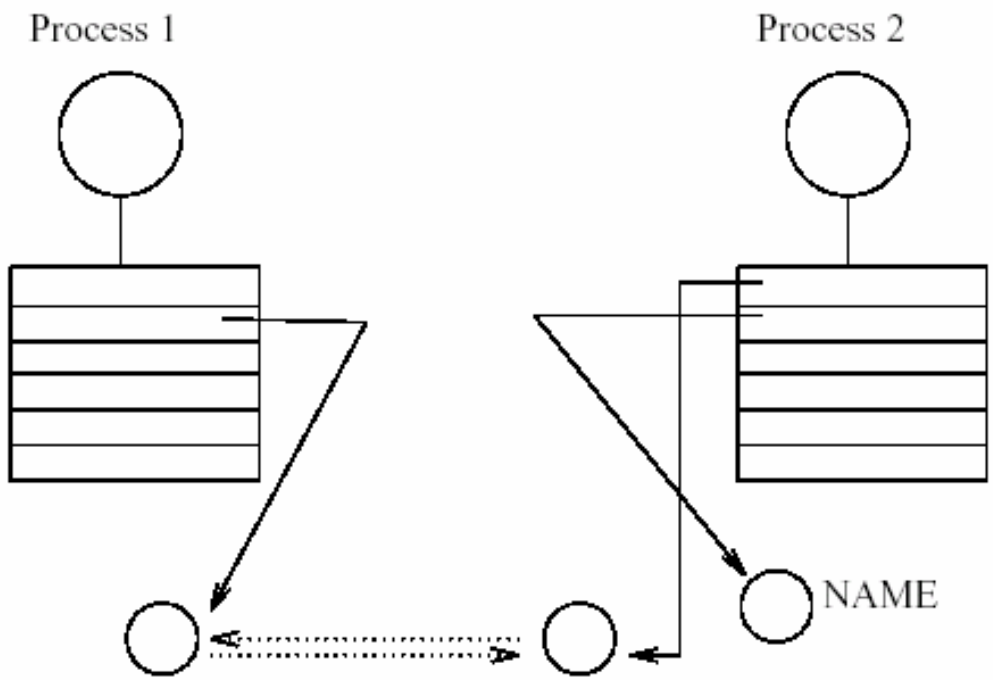
Tal como nos pipes:

- quando o último ficheiro escritor fecha o fifo é gerado um fim de ficheiro para o leitor do fifo
- se escrevermos para um ficheiro sem nenhum processo com ele aberto, é gerado o sinal SIGPIPE

IPCs entre diferentes máquinas

Comunicação entre processos em diferentes máquinas

- Quando os processos existem em máquinas distintas
- Bidireccional



Berkley Sockets

Índice de tópicos

- Comunicação entre processos usando sockets
- Arquitectura Cliente Servidor
- Rede
- Hosts, Portos, interfaces e sockets
- Modelos de ligação
- Sockets
- Comparação
- Cenários
- Exemplo

Comunicação entre processos usando Sockets

- Msg, queues, pipes, memória partilhada, semáforos, ficheiros...
  - Permitem comunicação entre processos
  - Mas entre processos dentro da mesma máquina
- Como conseguir comunicação entre sistemas remotos?
  - Acesso FTP
  - Telnet
  - Máquinas ligadas em Cluster
  - HTTP
  - ICQ

R: usando FTP,Telnet, Cluster (maquinas com ligações dedicadas de alta velocidade, normalmente fibra optica)

Arquitectura Cliente-servidor

- Daemon corre num computador
  - Actua como servidor
  - Nome do programa termina com “d” (UNIX, Linux)
    - Ftpd / Httpd
- Servidor “escuta” a rede num *porto* pré-definido
  - Normalização de *portos* para os protocolos mais comuns
    - Ficheiro /etc/services (Unix,linux), ou C:\winnt\System32\drivers\etc\services
  - *Porto* representa um identificador do processo que escuta a rede
  - Juntamente com o IP da máquina controi um *chave única*
  - Processo nível utilizador devem usar portos > 1024
  - Os restantes (<1024) só para processos com privilégios *superuser*

Arquitectura Cliente-servidor

- Responsabilidades do servidor
  - Segurança
  - Atender pedidos de clientes
  - Gerir a transmissão dos dados
  - Resolver os pedidos de forma apropriada
- Programa cliente
  - Liga ao servidor
  - Pede dados
  - Fornece a informação ao utilizador

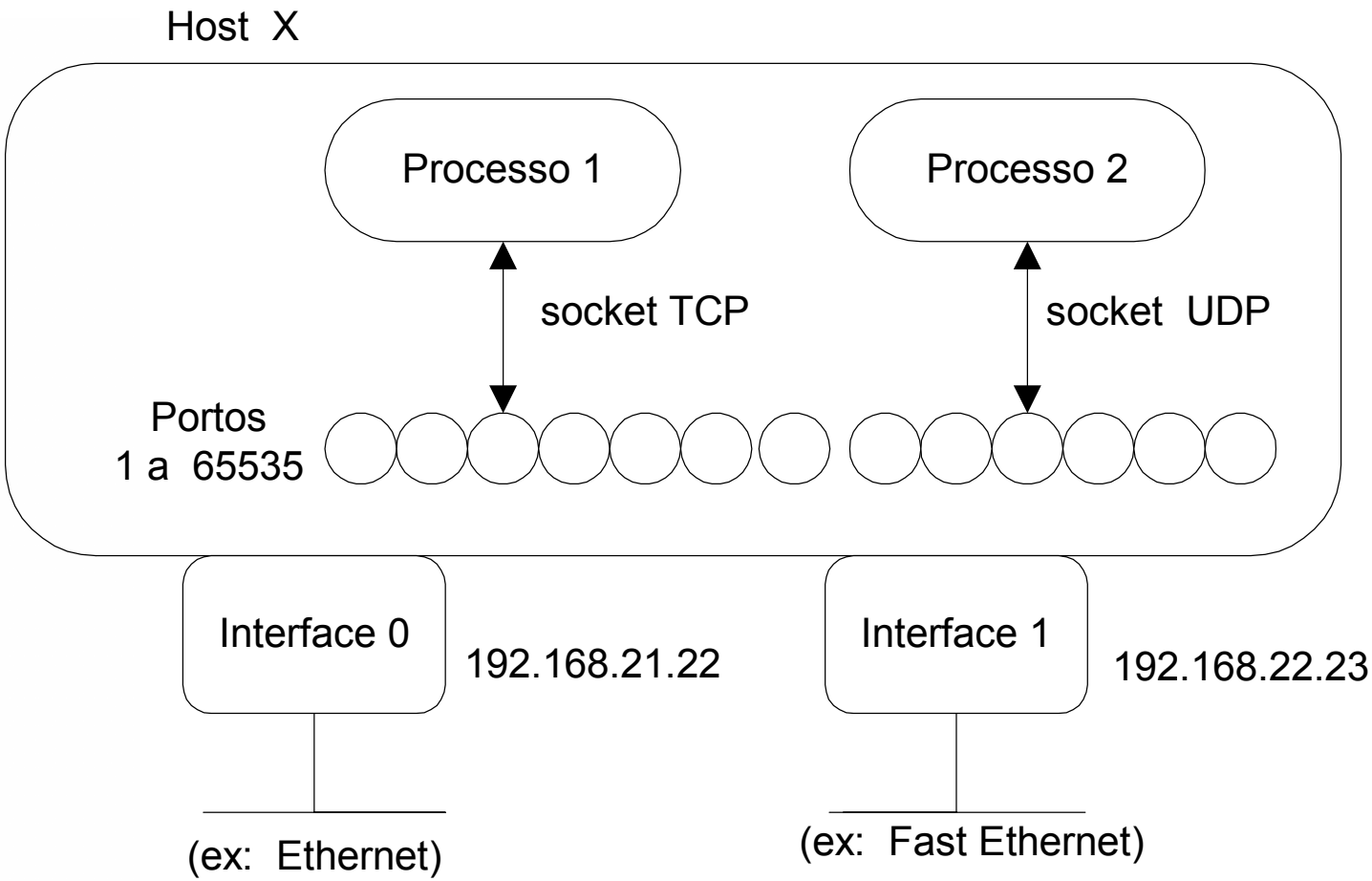
Rede

- Permite ligação entre duas máquinas remotas
  - Precisa de endereço IP
  - Identifica univocamente cada sistema
- Permite partilha de informação
- Permite cooperação entre processos

Rede

- Dois modelos de ligação
  - Orientado à conecção (*Connection Oriented)*
    - TCP (transmission Control Protocol)
      - FTP
      - Telnet
  - *Conectionless Oriented*
    - UDP (User Datagram Protocol)
      - HTTP

Hosts, Portos, Interfaces e Sockets



Modelos de ligação

Orientado à ligação

- Circuitos virtuais
- Não interrompe a ligação
- Garante que toda a informação é entregue/recebida
  - Recepção da informação é confirmada
- Garante integridade da informação
  - Necessário quando se tem de garantir a validade da informação
  - Maior o peso na comunicação ⇨ ligação mais lenta
  - (ex: telefone via rede – *voice over IP*)

Modelos de ligação

Connectionless Oriented

- Pequenos pacotes de mensagens enviados para diferentes endereços
- Não tem ligação definida
- Não verifica integridade de dados ou confirmação de recepção
- Não pode garantir que foi recebido

Modelos de ligação

Connectionless Oriented

- Menos carga de gestão de dados ⇨ maior velocidade de comunicação
- Modelo usado pelo protocolo HTTP (a informação não é a principal preocupação)
- É da responsabilidade do cliente garantir que toda a informação foi recebida
  - Voltar a pedir informação que não recebeu

Sockets

Conceitos

- Socket – Ponto de ligação para comunicação
- Dois processos podem comunicar, criando sockets e enviando mensagens entre eles
- Tipo de socket é determinado por:

  < Domínio , Tipo , Protocolo >

  - Domínio : para se alcançar o socket remoto é necessário que tenha nome. O formato do nome é dependente do *Domínio da comunicação* ou *Address Family* a que o socket pertence
  - Tipo: Semântica da comunicação para sockets
  - Protocolo : qual o protocolo que suporta o socket

Sockets

Criação - Domínio

```
Socket_descriptor= socket(domain, type, protocol)
Int Socket_descriptor, domain , type, protocol
```

- Domínio de comunicação ou *Address Family*
  - Define o formato do endereço
  - Todas as operações posteriores vão interpretar o endereço dado de acordo com este formato
  - Tipos de formato definidos (em <sys/sockets.h>):
    - AF_UNIX (espaço de nomes UNIX)
    - AF_INET (Endereços internet DARPA)
    - AF_OSI, ...

Sockets
Criação

- Quando o socket é criado, não tem endereço
- É preciso ligar (*Bind*) o socket a um endereço
- O socket só tem nome quando lhe é explicitamente atribuido um endereço através do system call *bind()*

```
Status = bind(sock, address , addrlen);
int status;       /* 0 se sucesso, -1 caso contrário */
int sock;         /* descritor  */
struct address;   /* endereço de rede/máquina  */
int addrlen;      /* dimensão (bytes) do endereço */
```

Sockets

Criação – Domínio UNIX

- AF_UNIX – Espaço de nomes UNIX
  - Usa nomes de directorias
  - O socket é criado no sistema de ficheiros
  - Necessários cuidados adicionais quando se fecha o socket
  - Só permite comunicação entre processos dentro da mesma máquina

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

struct sockaddr_un {
        short  sun_family;   /* AF_UNIX */
        char   sun_path[108-4];/*path name*/
}
```

Sockets

Criação – Domínio Internet

- AF_INET – endereços internet DARPA
  - Contituido por 2 partes:
    - Endereço de máquina (nº rede + nº máquina)
    - Número do porto
  - É o endereço máquina que permite a processo em máquinas diferentes comunicarem
  - O *porto* define multiplos endereços dentro da mesma máquina

Sockets

Criação – Domínio Internet

```
#include<netinet/in.h>

struct sockaddr_in {
       short  sin_family;       /* AF_INET */
       u_shortsin_port;         /* port number */
       struct in_addr   sin_addr /*net&host address(32 bits)*/
       char   sin_zero[8];    /*not used */
}

struct in_addr {
    union {
       struct {u_char s_b1, s_b2, s_b3, s_b4;} S_un_b;
       struct {u_char s_w1, s_w2;} S_un_w;
       u_long S_addr;
    } S_un;
#define s_addr S_un.S_addr /* used for most TCP&IP code */
}
```

Sockets

Criação - Tipo

- Tipo ou *style*
  - Definido em <sys/sockets.h>
    - SOCK_STREAM
    - SOCK_DGRAM
    - SOCL_RAW
    - SOCK_RMD (Reliable Datagram)
    - SOCK_SQPACKET (Sequence packet Stream)
  - Cada um suportado por diferentes protocolos

Sockets

Criação - Tipo

- SOCK_DGRAM
  - Modelo de comunicação *connectionless*

    Mensagens independentes (sem relação de ordem e normalmente pequenas) designadas por datagrams. São enviadas pela camada de transporte para um dado endereço. Podem-se perder, aparecer duplicadas ou fora de ordem
  - Usa o protocolo UDP da camada de transporte

```
Sock = socket ( AF_INET, SOCK_DGRAM, 0 )
```

Sockets

Criação - Tipo

- SOCK_STREAM
  - Serviço fiável e orientado à conecção (*connection oriented*)
  - Informação transmitida em full-duplex e controlo de fluxo
  - Informação entregue ordenada, sem perdas, erros ou duplicação. Caso contrário, aborta a ligação e devolve erro
  - É suportado pelo protocolo TCP

```
Sock = socket ( AF_INET,SOCK_STREAM, 0 )
```

Sockets

Criação - Protocolo

- UDP (User datagram protocol) - suporta SOCK_DGRAM
- TCP (Transmission Control Protocol) - suporta SOCK_STREAM
- IP (Internet Protocol)

- Valores que podem ser usados:
  - Se “ 0 “ assume o protocolo por omissão para a família e tipo de socket
  - Caso contrário, usa os valores definidos em <netinet/in.h>

Sockets

Estabelecer ligação – Stream sockets

- Tem de estabelecer ligação antes de transferir dados
- Um socket pode ser *activo* ou *passivo*. Após criação é *activo.* Fica *passivo* depois de chamar *Listen()* e *accept()*.
- Só sockets *activos* podem ser usados no system call *connect()*.
- Só sockets *passivos* podem ser usados em *accept()*
- Estabelecem ligação via *connect()*

Sockets

Estabelecer ligação – Stream sockets

- Quando o socket é criado, não tem endereço
- É preciso ligar (*Bind*) o socket a um endereço
- O socket só tem nome quando lhe é explicitamente atribuido um endereço através do system call *bind()*

```
Status = bind(sock, address , addrlen)
Int status;       /* 0  se sucesso, -1 caso contrário */
Int sock;         /* descritor  */
Struct addr  * address; /*endereço de rede/máquina  */
Int addrlen;    /* dimensão (bytes) do endereço */
```

Sockets

Estabelecer ligação – Stream sockets

```
status = listen( sock, queuelen)
int    sock, queuelen;

new_socket = accept (old_sock, name, namelen)
int    new_sock,old_sock; /* socket descriptors */
struct sockaddr  *name;   /*peer socket on new conection */
int    *namelen;          /*length of name (bytes) */
```

- *Listen* – define um buffer de dimensão *queuelen*
- *Accept* – pega na 1ª ligação pendente do buffer e retorna um novo (e já ligado) socket
- *O socket antigo mantêm-se e pode ser novamente usado*
- *Se não existir mais ligações pendentes, accept() bloqueia*

Sockets

Estabelecer ligação – DGRAM sockets

```
Status = connect (sock, name, namelen )
Int sock;
Struct sockaddr *namelen;
Int namelen;
```

- Um socket *activo* liga-se a um socket *passivo*, através da função *connect( )*
- *Name* é o endereço do socket remoto, de acordo com o domínio de comunicação definido na criação do socket

Sockets

Criação - Transferência de dados

- Usa os *system Calls*
  - read / write - recvmsg / sendmsg
  - recv / send - readv / writev
  - recvfrom / sendto
- Podem ser todos usados para transferir dados através de sockets
- O mais apropriado, depende depende da funcionalidade:
  - send /recv - tipicamente para SOCK_STREAM
  - Sendto / recvform - tipicamente para SOCK_DGRAM
  - Read / write - para qualquer socket já criado

Sockets

Terminar ligação

- System call *close ()*

```
status = close (sock)
int    status,sock;
```

- System call *unlink ()* – Só para AF_UNIX e depois de fechar o socket

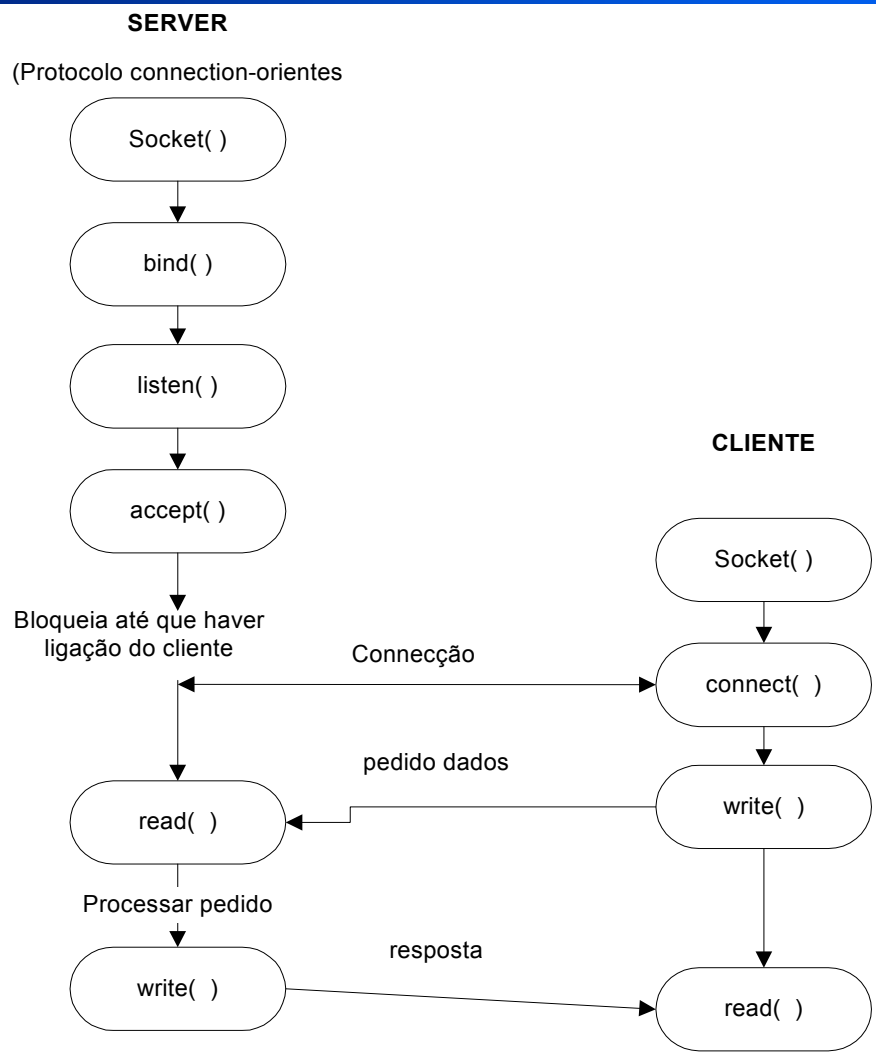
```
status = unlink (pathname)
int    status;
char *pathname
```

Comparação

- Servidores
  - Ambos abrem socket e fazem *bind* com o endereço local
  - Modelo orientado à conecção
    - Servidor tem de fazer *listen()* e *accept( )*
- Clientes
  - Ambos devem abrir o socket
  - Modelo orientado à conecção
    - Tem de fazer *connect()* ao servidor
  - *Connectionless model*
    - Deve ligar (*bind) socket e endereço local*

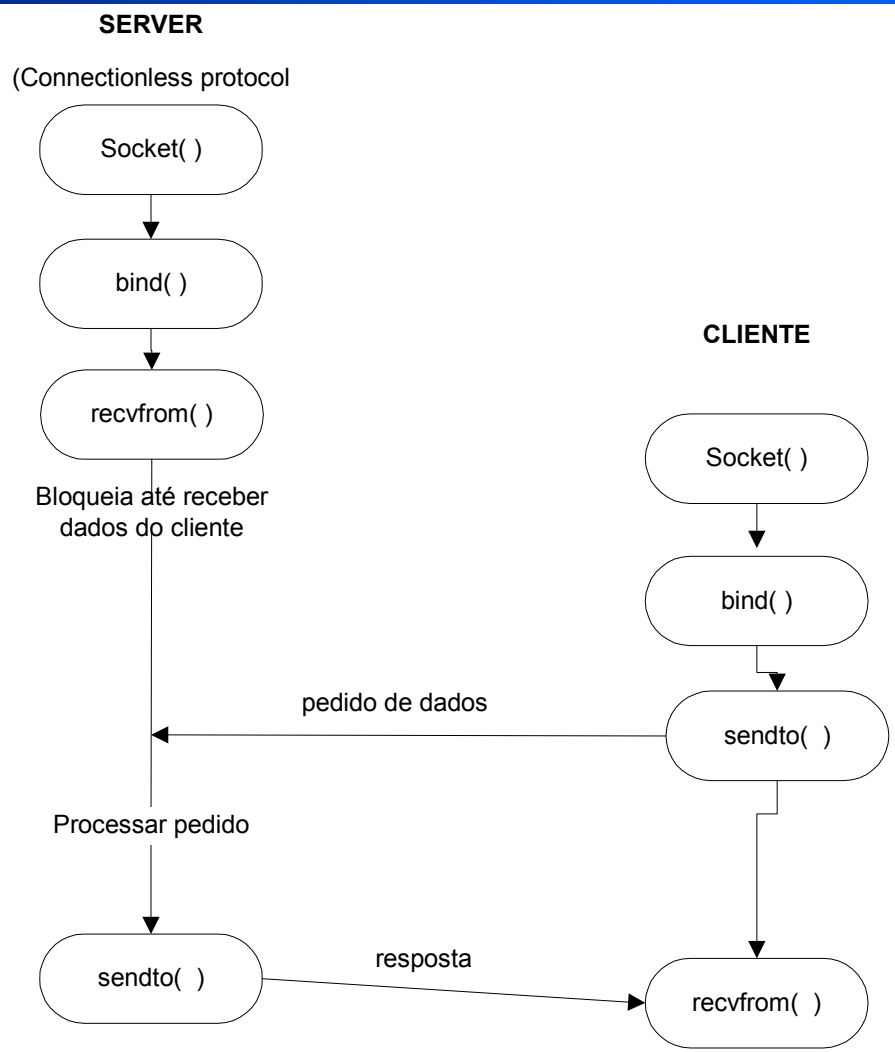
Cenário

Estabelecer ligação orientada à connecção



Cenário

Estabelecer ligação *connetionless oriented*



Exemplo
Servidor

```
#include<ctype.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet.h>
#include<signal.h>
#define SIZE sizeof (struct sockaddr_in);

Void catchint(int sig);
Int newsockfd;

Main ( ) {
    int sockfd;
    char c;
    struct sockaddr_in sever=(AF_INET,7000,INADDR_ANY);

    if ((sockfd=socket (AF_INET,SOCK_STREAM,0))==-1) {
        perror("Socket Call Failed\n");
        exit(1);
    }
```

Exemplo
Servidor

```
if bind(sockfd, (struct sockaddr *) &server,SIZE)==-1) {
    perror("Bind Call Failed\n");
    exit(1);
}
if(listen(sockfd,5)==-1) {
    perror("Listen Call Failed\n");
    exit(1);
}
for(;;) {
    if ((newsockfd=accept(sockfd,NULL,NULL))== -1) {
        perror("Accept Call Failed\n");
    }
    if (fork( )==0) {
        While (recv(newsockfd,&c,1,0) >0 ){
            c=toupper(c);
            send(newsockfd,&c,1,0);
        }
        close(newsockfd);
        exit(0);
    }
}
```

Exemplo

Cliente

```
(...)
Main ( ) {
    int sockfd;
    char c,rc;
    struct sockaddr_in server=(AF_INET,7000);

    Server.sin_addr.s_addr=inet_addr("197.45.10.2");
    if ((sockfd=socket (AF_INET,SOCK_STREAM,0))==-1) {
        perror("Socket Call Failed\n");
        exit(1);
    }
    if(connect(sockfd, (struct sockaddr *) &server,SIZE)==-1) {
        perror("Bind Call Failed\n");
        exit(1);
    }
```

Exemplo
Cliente

```
        for (rc='\n';;))) {
            if (rc=='\n')
                    printf("Inserir letras minúsculas\n");

            c=getchar( );
            send((sockfd,&c,1,0);

            if (recv(sockfd,&rc,1,0)>0)
                printf("%c",rc);
            else {
                printf("Servidor terminou a execução\n");
                close(sockfd);
                exit(1);
            }
        }

    }
```