sockets
interprocess communication

Taisy Weber

Comunicação entre processos

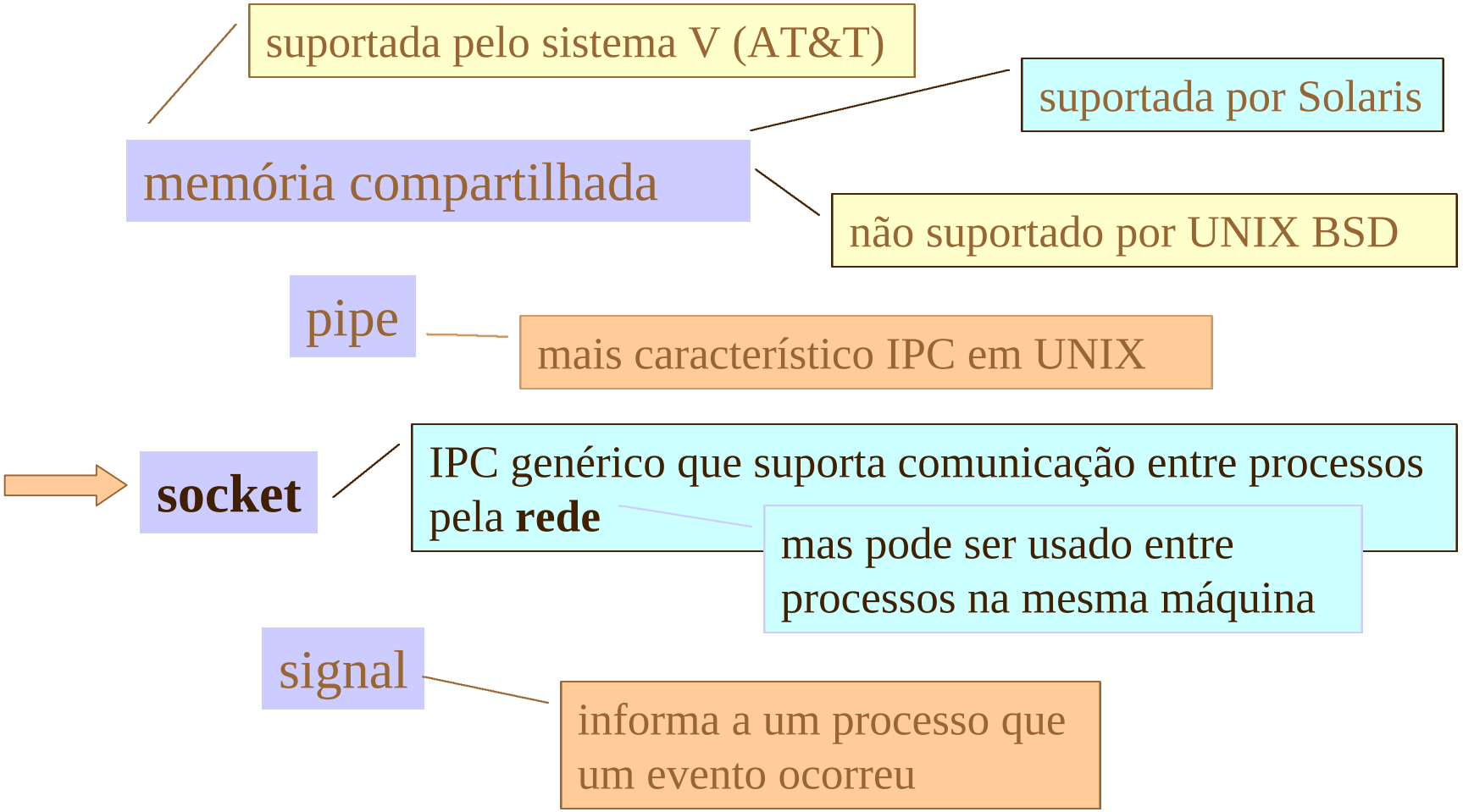
- ✓ Mecanismos
  - Pipes, FIFO (named pipes), semáforos, message queues.
  - Memória compartilhada.
  - Sockets
    - Definição, chamadas de sistemas, utilização de sockets.

Mitchell, cap 5

Johnson, cap 16

Matthew & Stones, cap 14

IPC em UNIX


suportada pelo sistema V (AT&T)
suportada por Solaris
memória compartilhada
não suportado por UNIX BSD
pipe
mais característico IPC em UNIX
socket
IPC genérico que suporta comunicação entre processos pela **rede**
mas pode ser usado entre processos na mesma máquina
signal
informa a um processo que um evento ocorreu

Exemplos de IPC

- ✓ pipes
  - mais característico mecanismo IPC em UNIX
  - a mais antiga forma de IPC
  - usados na shell
- ✓ sockets
  - mecanismo genérico de IPC que suporta comunicação entre processos pela **rede** (mas pode ser usado também para comunicação de processos na mesma máquina)

UNIX sockets

- ✓ socket é um ponto de comunicação
  - comunicação por sockets pode ser usada por processos não relacionados

pipes apenas entre pais e filhos

  - um socket em uso tem um endereço associado

a natureza do endereço depende do **domínio** da comunicação

exemplos: serviços (aplicações cliente servidor) `rlogin`, `httpd` e `ftp` usam sockets no **domínio** Internet

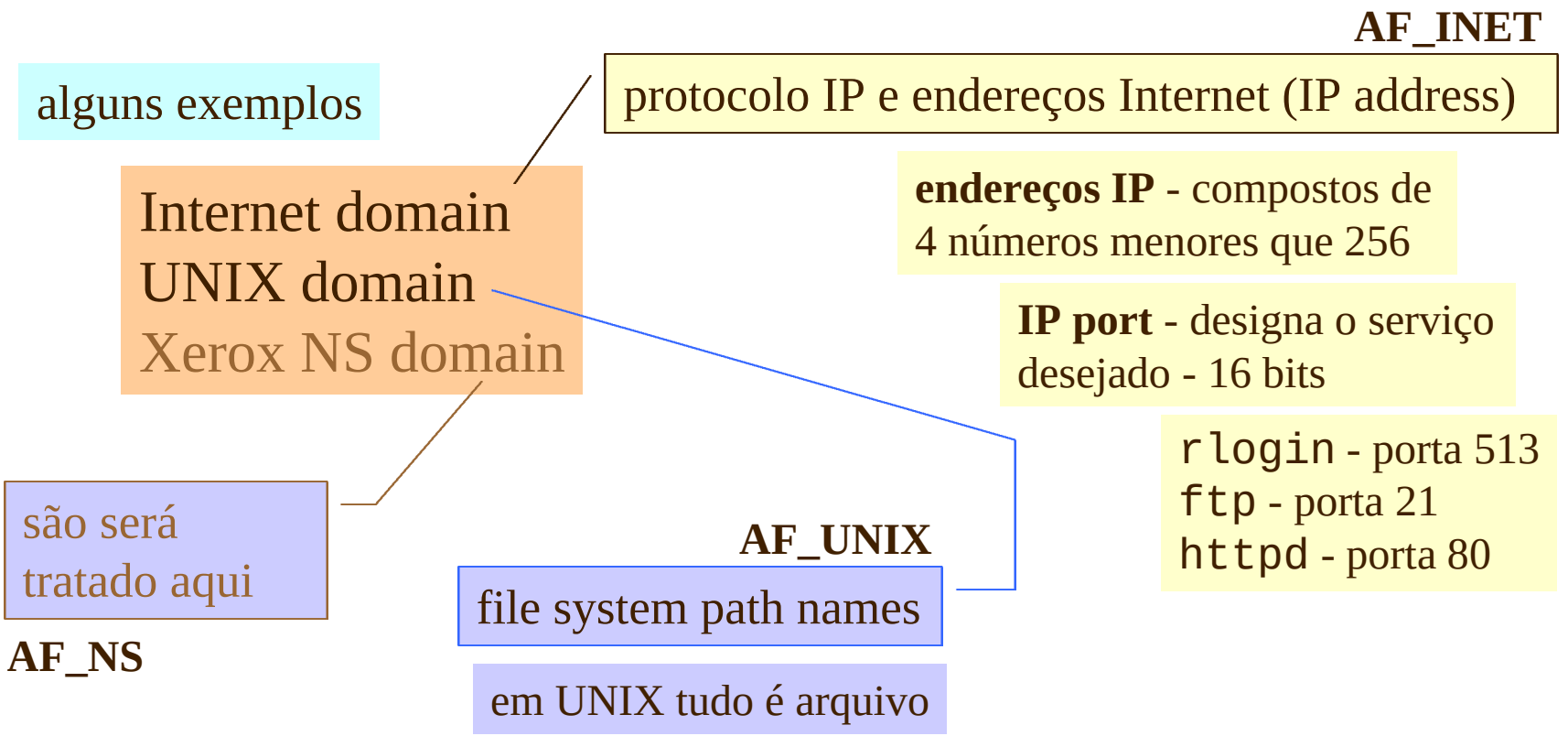
Atributos de sockets

- ✓ sockets são caracterizados por
  - domínio
  - tipo
  - protocolo
- ✓ um socket também tem um nome
  - endereço associado ao socket

a natureza do endereço depende do domínio

Domínio

- ✓ especifica a rede de comunicação que será usada



Domínios Internet e UNIX

- ✓ Internet
  - endereço especifica a máquina (IP address) e o serviço na máquina (IP port)
  - serviços padrões possuem endereços padrões
- ✓ UNIX
  - usado por processos em uma única máquina
    - **protocolo**: entrada e saída de arquivos
    - **endereço**: nomes de arquivos (absoluto)

Tipos de sockets

- ✓ tipo representa uma classe de serviços

  se o tipo é implementado em dado domínio, podem exitir um ou mais protocolos que o implementam

  - stream sockets
  - sequenced packet sockets
  - datagram sockets
  - reliable delivered message sockets
  - raw sockets

stream sockets

TIPO: `SOCK_STREAM`

- ✓ fluxo duplex de dados — confiável e seqüencial
- ✓ nenhum dado é perdido
- ✓ nenhum dado é duplicado

comunicação confiável

- ✓ sem limites para tamanho de mensagens

no **domínio Internet** suportado pelo protocolo TCP/IP

no **domínio Unix**, arquivos

nos exemplos vamos nos deter nesse tipo de socket apenas

sequenced packet sockets

- ✓ fluxo duplex de dados

confiável e seqüencial

- ✓ nenhum dado é perdido
- ✓ nenhum dado é duplicado

comunicação confiável

- ✓ **com** limites para tamanho de mensagem

única diferença para stream sockets

usado no **domínio Xerox NS**

datagram sockets

TIPO: `SOCK_DGRAM`

não confiável e **sem** garantia de ordenação

- ✓ bidirecional
  - dados podem ser perdidos
  - dados podem ser duplicados

  } comunicação **não** confiável
- ✓ tamanho variável de mensagens
  - mas o tamanho da mensagem original é preservado na recepção
- ✓ não orientada a conexão

cada mensagem carrega seu endereço

no **domínio Internet** suportado pelo protocolo UDP/IP

reliable delivered message sockets

- ✓ parecido com datagram sockets
  - não orientado a conexão
  - tamanho variável de mensagem
- ✓ mas com garantia de entrega

por isso é dito **confiável** *(reliable)*

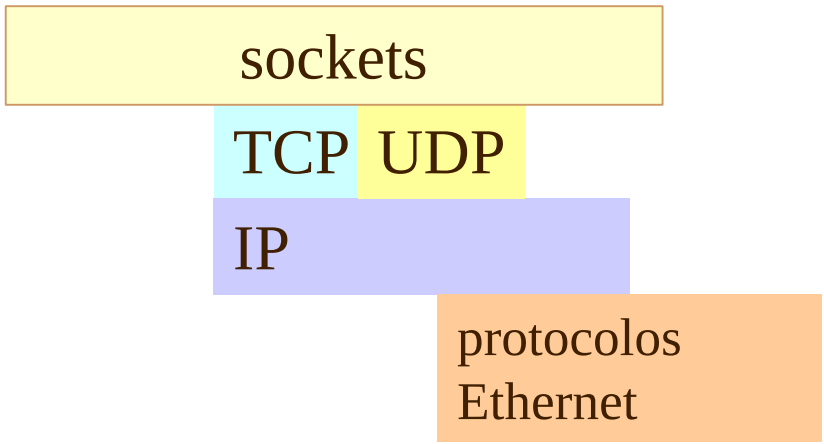
de acordo com livro texto ainda não suportado por nenhum sistema

Johnson, cap 16

raw sockets

soquete bruto

– permite acesso direto a protocolos de comunicação de baixo nível

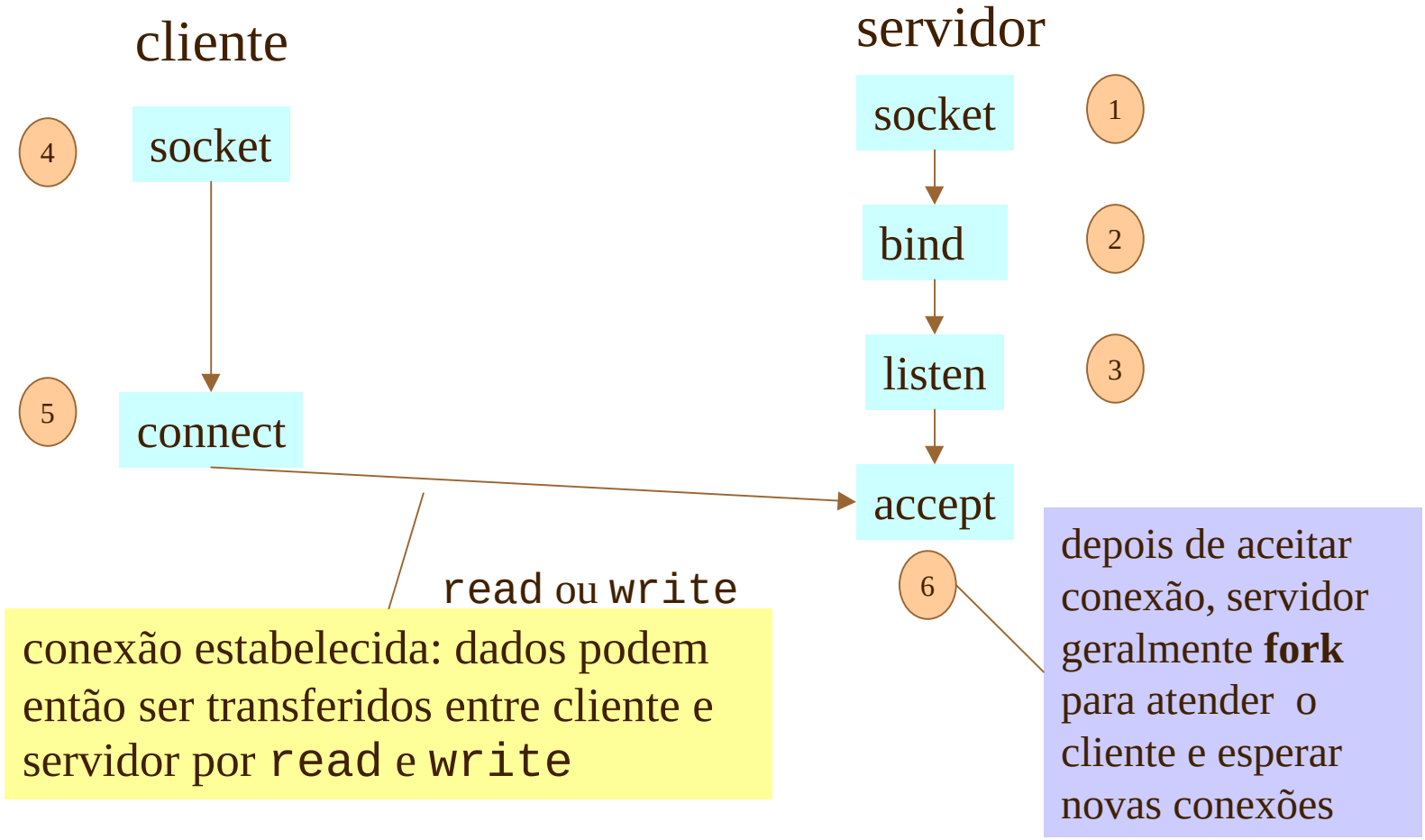


útil para o **desenvolvimento de novos protocolos** de comunicação em redes

considerado por alguns como furo de segurança

Cliente & servidor

exemplo: *stream socket*



Syscalls para stream sockets

usada pelo cliente e pelo servidor

- socket — cria um socket,
  argumentos: domínio, tipo e protocolo

servidor

- bind — servidor dá **nome** a um socket para que outro processo possa usá-lo

  o nome, ou endereço, é bem conhecido dos possíveis clientes
- listen — servidor indica ao kernel que está apto a receber conexões de clientes
- accept — servidor aceita conexão

Mais syscalls para stream sockets

usado por processo cliente

– connect

inicia uma conexão

liga dois sockets:
um local e um externo ao processo

antes da conexão processo cliente deve **criar** o seu socket local usando a *system call* **socket( )**

Usando stream sockets

modelo cliente servidor

- para estabelecer conexão

  socket, bind, listen, accept e fork

- após estabelecer conexão

  system calls **read** e **write** comuns para transferir dados

- para terminar a conexão

  close - destroi o socket

  shutdown - termina apenas uma direção em uma conexão duplex

Syscalls para datagramas

- ✓ para datagramas

que operam sem conexão

argumentos:

sendto

recvfrom

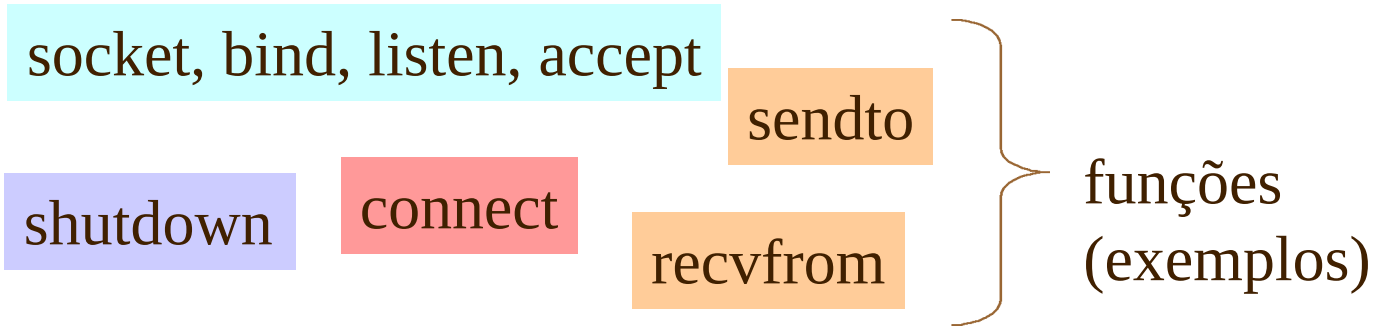
descritor do socket
pointer e tamanho de um *buffer* de mensagem
pointer e tamanho de um endereço

endereço para envio ou o endereço do qual foi recebida mensagem

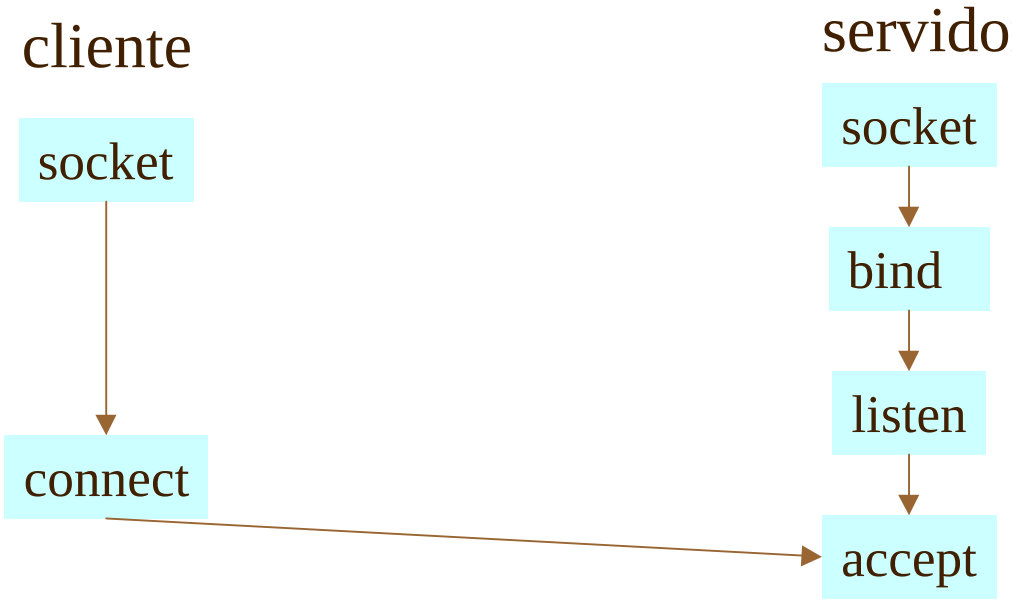
System call *socketcall*

- Linux possui apenas uma *system call* para sockets
  - toda a programação com sockets é feita através de **socketcall**

o parâmetro *call* indica qual a função



Exemplo 1: cliente & servidor



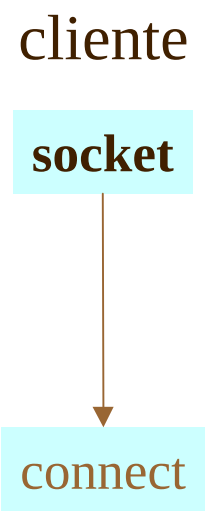
fontes: `client1` e `server1`

Matthew & Stones, cap 14

stream socket no domínio Unix: AF_UNIX

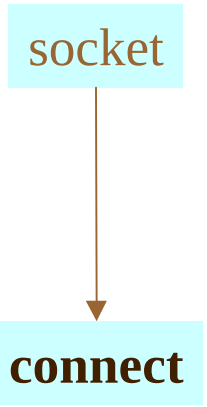
Exemplo 1: cliente ...

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>

int main()
{
    int sockfd;
    int len;
    struct sockaddr_un address;
    int result;
    char ch = 'A';

/*  Create a socket for the client.  */

    sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
```



`socket` cria um socket sem dar nome

Exemplo 1: cliente



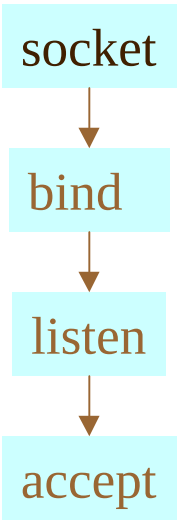

```
/*  Name the socket, as agreed with the server.  */
    address.sun_family = AF_UNIX;
    strcpy(address.sun_path, "server_socket");
    len = sizeof(address);

/*  Now connect our socket to the server's socket.  */
    result = connect(sockfd, (struct sockaddr *)&address, len);

    if(result == -1) {
        perror("oops: client1");
        exit(1);
    }

/*  We can now read/write via sockfd.  */
    write(sockfd, &ch, 1);
    read(sockfd, &ch, 1);
    printf("char from server = %c\n", ch);
    close(sockfd);
    exit(0);
}
```

`connect` conecta com um socket no servidor chamado `server_socket`

significa: endereço (arquivo) é `server_socket`

se tentar executar esse programa sem o server, o programa falha

Exemplo 1: servidor ...

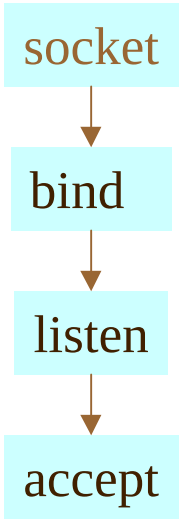



```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>

int main()
{
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    struct sockaddr_un server_address;
    struct sockaddr_un client_address;

/*  Remove any old socket and create an unnamed socket for the server.  */

    unlink("server_socket");
    server_sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
```

domínio Unix

Exemplo 1: servidor ...

```
/*  Name the socket.  */

    server_address.sun_family = AF_UNIX;
    strcpy(server_address.sun_path, "server_socket");
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);

/*  Create a connection queue and wait for clients.  */

    listen(server_sockfd, 5);
    while(1) {
        char ch;
        printf("server waiting\n");

/*  Accept a connection.  */

        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd,
            (struct sockaddr *)&client_address, &client_len);
```



Exemplo 1: servidor

```
/*  We can now read/write to client on client_sockfd.  */

        read(client_sockfd, &ch, 1);
        ch++;
        write(client_sockfd, &ch, 1);
        close(client_sockfd);
    }
}
```

o programa pode servir apenas um cliente de cada vez

1. executar o programa servidor em *background* para que possa rodar independentemente
2. verificar a existência do socket com o comando `ls`
3. rodar o cliente em *foreground*

socket

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

domain: `AF_INET` `AF_UNIX` `AF_NS`

type: `SOCK_DGRAM` `SOCK_STREAM`

protocol: usado quando existe escolha (0 seleciona protocolo default)

a função retorna um descritor do socket (muito semelhante a um descritor de arquivo)

do ponto de vista do programador, todos os sockets atuam como **descritores de arquivos** e são endereçados por um valor inteiro

endereços domínios INET e UNIX

cada domínio requer seu próprio formato de endereço

definida em `sys/un.h`

```
struct sockaddr_un  {
          sa_family_t       sun_family ;   /* AF_UNIX */
          char              sun_path[ ];   /* pathname */
```

pathname limitado em Linux a 108 caracteres

definida em `netinet/in.h`

```
struct sockaddr_in  {
          short int         sin_family; /* AF_INET
          unsigned short int sin_port;   /* port number
          struct in_addr    sin_addr;    /* Internet address of the host machine
};
```

estrutura de endereço IP

```
 struct in_addr{
           unsigned long int   s_addr;
 };
```

bind

lado do servidor

- o socket recebe um nome para se tornar disponível para uso por outros processos

```
#include <sys/socket.h>
int bind(int socket, const struct sockaddr *address, size_t address_len)
```

`address_len` tamanho da estrutura `address`

tamanho e formato do endereço depende da família

`AF_INET` IP port number

`AF_UNIX` nome de arquivo (path)

listen

lado do servidor

```
#include <sys/socket.h>
int listen (int socket, int backlog);
```

- cria uma fila para armazenar requisições de conexão pendentes

`backlog` tamanho da lista, usualmente 5

conexões até esse limite ficam esperando na fila
acima do limite serão recusadas

accept

lado do servidor

```
int accept(int socket, struct sockaddr *address, size_t *address_len);
```

- retorna quando um cliente tenta se conectar ao socket `socket`
- accept cria um novo socket para se comunicar com o cliente e retorna seu descritor
  - o socket tem o mesmo tipo do socket de listen
  - endereço do cliente: `sockaddr`
    - » se não for interessante usar pointer nulo
- bloqueio

  usar flag **nonblock** se desejar comportamento diferente
  - accept **bloqueia** se não há nenhuma conexão pendente na fila (listen)

lado do cliente

connect

```
int connect(int socket, const struct sockaddr *address, size_t address_len)
```

- estabelece uma conexão entre o socket criado no cliente (sem nome, apenas com descritor) com um socket de servidor (nomeado com bind)
  - socket do servidor endereçado por `address`
  - socket do cliente deve ter sido criado e ter recebido um descritor válido

se a conexão não puder ocorrer imediatamente, connect bloqueia

tb. pode ser alterada

pode sair por timeout
pode ser interrompida por um signal a ser tratado

close

- ✓ igual ao velho close usado em sistemas de arquivos
  - – funções de baixo nível

Exemplo com network socket

- ✓ o livro continua o capítulo com
  - exemplo de socket de rede
  - discussão sobre múltiplos clientes
    - ou seja múltiplas conexões simultâneas
      - uso de fork
      - listen continua válido
      - o socket criado é herdado pelo filho

Fim

- de sockets ...