


© 2004-2015 Volnys Bernal 1

## Servidor UDP


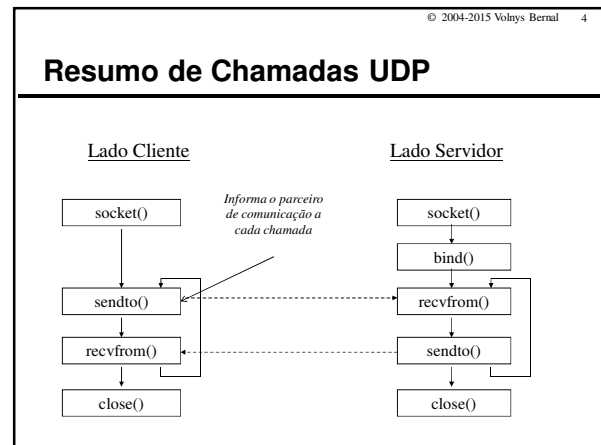
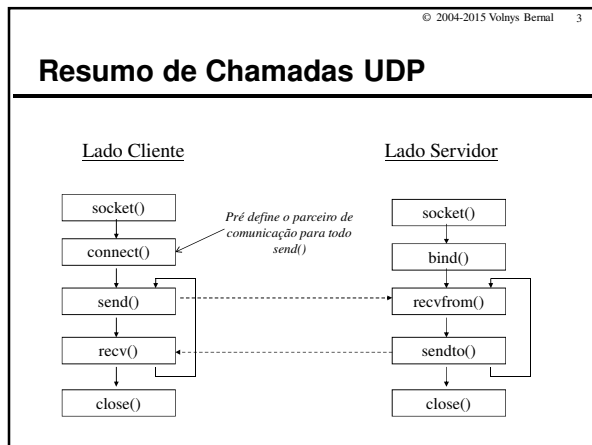
**Volnys Borges Bernal**  
volnys@lsi.usp.br

Departamento de Sistemas Eletrônicos  
Escola Politécnica da USP




© 2004-2015 Volnys Bernal 2

## Resumo das Chamadas UDP

© 2004-2015 Volnys Bernal 5

## Chamada socket()



© 2004-2015 Volnys Bernal 6

## Chamada socket()

- ❑ **Objetivo**
  - ✦ Criar um novo socket (plug de comunicação)
- ❑ **Resultado**
  - ✦ Retorna o descritor de arquivo (número inteiro).
- ❑ **Sintaxe**

```
sd = socket (int domain, int type, int protocol)
```
- ❑ **Observação:**
  - ✦ Quando um socket é criado, não possui nenhuma informação sobre o parsocket (endereços IPs e portas dos parceiros).
  - ✦ Os endereços IP e portas são informados nas chamadas:
    - Lado servidor: bind()
    - Lado cliente: connect() ou sendto()

© 2004-2015 Volnys Bernal 7

## Chamada socket()

❑ **Sintaxe geral**

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

*Socket descriptor*

*Id. do protocolo:*

- UDP (17)
- TCP (6)

*Pilha de protocolos:*

- PF\_LOCAL (file)
- PF\_INET (IPv4)
- PF\_INET6 (IPv6)
- PF\_X25 (X25)

*Tipo da comunicação:*

- SOCK\_STREAM (TCP)
- SOCK\_DGRAM (UDP)
- SOCK\_RAW (IP)

© 2004-2015 Volnys Bernal 8

## Chamada socket()

❑ **Tipo de serviço**

- ❖ SOCK\_STREAM
  - Para ser utilizado com o protocolo TCP
  - Canal de comunicação full duplex
  - Orientada a conexão
  - Fluxo de bytes sem delimitação
  - Comunicação confiável: sem perda de dados, sem duplicação, entrega na ordem
  - Chamadas para transmissão e recepção de dados:
    - read(), write() ou send(), recv()
- ❖ SOCK\_DGRAM
  - Para ser utilizado com o protocolo UDP
  - Datagrama (mensagens)
  - Chamadas para transmissão e recepção de dados:
    - send(), recv()
- ❖ SOCK\_RAW
  - Permite acesso a protocolos de mais baixo nível
  - Datagrama (mensagens)
  - Chamadas para transmissão e recepção de dados:
    - send(), recv()

© 2004-2015 Volnys Bernal 9

## Chamada socket()

❑ **Para criar um socket TCP**

```
#include <sys/socket.h>
sd = socket(AF_INET, SOCK_STREAM, 0);
```

❑ **Para criar um socket UDP**

```
#include <sys/socket.h>
sd = socket(AF_INET, SOCK_DGRAM, 0);
```

❑ **Para criar um socket Raw**

```
#include <sys/socket.h>
sd = socket(AF_INET, SOCK_RAW, protocol);
```

© 2004-2015 Volnys Bernal 10

## Chamada socket()


❑ **Exemplo de criação de socket UDP**

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

int sd; // socket descriptor
...
sd = socket(PF_INET, SOCK_DGRAM, 0);
if (sd == -1)
{
    perror("Erro na chamada socket");
    exit(1);
}
...
```

© 2004-2015 Volnys Bernal 11

## Chamada bind()



© 2004-2015 Volnys Bernal 12

## Chamada bind()

❑ **Objetivo**

- ❖ Permite associar um *socket address* (IP+porta) ao socket
- ❖ Deve ser utilizado no lado servidor

❑ **Resultado**

- ❖ Retorna -1 no caso de erro

© 2004-2015 Volnys Bernal 13

## Chamada bind()

❑ Sintaxe

```
#include <netdb.h>
```

```
int bind(
    int sd,
    struct sockaddr *myaddr,
    socklen_t addrlen)
```

Resultado da chamada: sucesso ou erro

Endereço IP da interface local.  
Pode ser o endereço IP de:

- Uma interface específica
- Todas interfaces locais (INADDR\_ANY)

Descritor do socket

Tamanho da estrutura de endereço (sockaddr)

© 2004-2015 Volnys Bernal 14

## Chamada bind()

```
#include <netdb.h>
```

```
struct sockaddr_in mylocal_addr
. . .
```


```
mylocal_addr.sin_family      = AF_INET;
mylocal_addr.sin_addr.s_addr = INADDR_ANY;
mylocal_addr.sin_port        = htons(myport);
```

```
status = bind (sd,
    (struct sockaddr *) &mylocal_addr,
    sizeof(struct sockaddr_in));
```

```
if (status == -1)
    perror("Erro na chamada bind");
```

© 2004-2015 Volnys Bernal 15

## Chamada recvfrom()



© 2004-2015 Volnys Bernal 16

## Chamada recvfrom()

❑ Recebimento de datagramas

❑ Permite obter a identificação do parceiro

❑ Pode ser utilizado pelo cliente ou servidor

```
int recvfrom(int sd, void * buffer, int buffersize, int flags,
    struct sockaddr *from, int *sockaddrsz)
```

Socket Descriptor

Ponteiro para buffer da mensagem

Tamanho do buffer

Endereço da origem

Tamanho da estrutura sockaddr

Opções

© 2004-2015 Volnys Bernal 17

## Chamada recvfrom()

❑ Exemplo:

```
char rxbuffer[80];
struct sockaddr_in fromaddr; // sockaddr do cliente
int size;

. . .
size = sizeof(struct sockaddr_in);
status = recvfrom(sd, rxbuffer, sizeof(rxbuffer), 0,
    (struct sockaddr *) &fromaddr,
    &size);
```

```
if (status <= 0)
    perror("Erro na chamada recvfrom");
```

```
. . .
```

© 2004-2015 Volnys Bernal 18

## Chamada recvfrom()

❑ Bloqueante

- ❖ Se não existirem mensagens na fila de recepção o processo fica aguardando sua chegada
- ❖ Exceção: quando o socket for criado como não bloqueante (ver fcntl(2)).

❑ Retorno

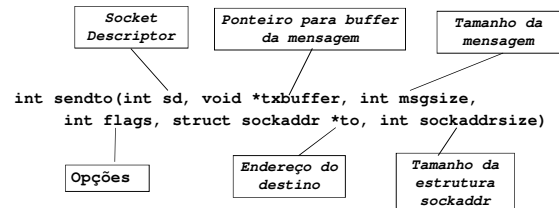
- ❖ Se a chamada tiver sucesso, o valor retornado é o tamanho do datagrama

## Chamada sendto()



## Chamada sendto()

- ❑ Função para transmissão de dados (datagramas)
- ❑ Para um destino especificado
- ❑ Não deve ser precedido por connect()
- ❑ Pode ser utilizado pelo cliente ou pelo servidor



## Chamada sendto()

### ❑ Exemplo:

```
...
status = sendto (sd, buffer, strlen(buffer)+1,
                0,
                (struct sockaddr *) &fromaddr,
                sizeof(struct sockaddr))

if (status <= 0)
    perror("Erro na chamada sendto");
...
```

## Chamada close()



## Chamada close()

### ❑ Objetivo

- ✦ Fechar o socket.
- ✦ Se ainda existirem dados para serem transmitidos pelo socket, aguarda por alguns segundos a finalização desta transmissão.

### ❑ Resultado

- ✦ Fecha o descritor do arquivo.

### ❑ Sintaxe

```
int socket (int sd)
```

## Chamada close()


### ❑ Exemplo:

```
int sd; // socketdescriptor
...

status = close(sd);
if (status == -1)
    perror("Erro na chamada close");
...
```

© 2004-2015 Volnys Bernal 25

## Exercício



© 2004-2015 Volnys Bernal 26


## Exercício

(1) Implemente um servidor para o serviço “echo” utilizando o protocolo UDP.

- ❖ O serviço “UDP echo” responde exatamente com a sequência ASCII recebida.

© 2004-2015 Volnys Bernal 27

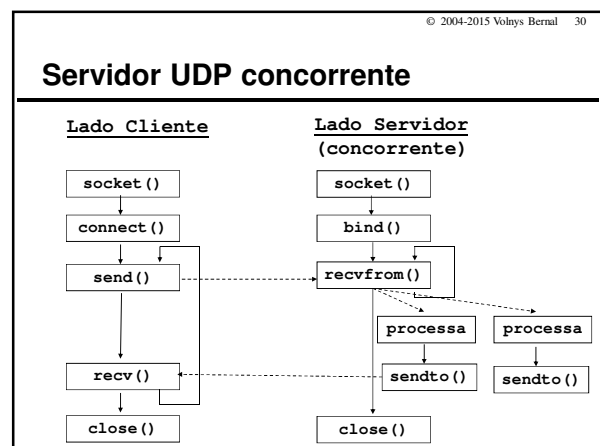
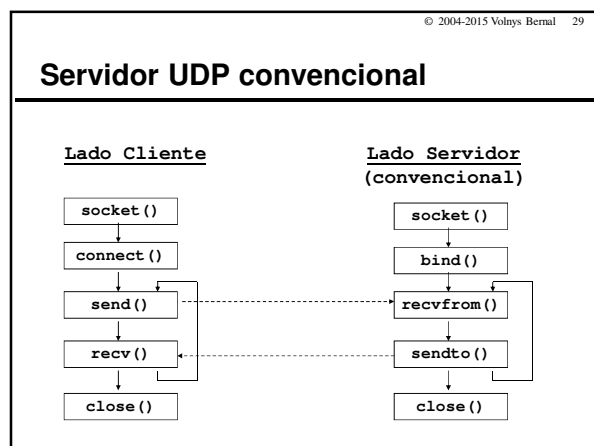
## Servidor Não Concorrente x Concorrente



© 2004-2015 Volnys Bernal 28

## Servidor não concorrente x concorrente

- ❑ **Servidor não concorrente**
  - ❖ Processa uma requisição por vez
- ❑ **Servidor concorrente**
  - ❖ Possui capacidade para processar várias requisições simultaneamente
  - ❖ Necessário quando o processamento da requisição for demorado e existir a possibilidade de chegar várias requisições simultâneas
  - ❖ Implementação mais complexa, com duas alternativas:
    - Uso de chamadas não bloqueantes
    - Uso de programação *multithreaded*



## Exercício



## Exercício

(3) Implementar um servidor UDP concorrente que realiza o seguinte processamento:

```
#include <time.h>

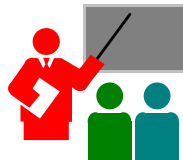
void processar(char* str_in, char* str_out)
{
    time_t now_time;
    struct tm now_tm;
    char str_inicio[40];
    char str_fim[40];

    now_time = time(NULL); // Obtém instante (data/hora) corrente
    now_tm = *localtime(&now_time); // Converte para hora local
    // Formata data/hora: "ddd yyyy-mm-dd hh:mm:ss zzz"
    strftime(str_inicio, sizeof(str_inicio), "%a %d/%m/%Y %H:%M:%S %Z", &now_tm);

    sleep(60);
    now_time = time(NULL); // Obtém instante (data/hora) corrente
    now_tm = *localtime(&now_time); // Converte para hora local
    strftime(str_fim, sizeof(str_fim), "%a %d/%m/%Y %H:%M:%S %Z", &now_tm);

    sprintf(str_out, "%s: Inicio: %s, fim: %s ", str_in, str_inicio, str_fim);
}
```

## Referências Bibliográficas



## Referências Bibliográficas

- ❑ **COMMER, DOUGLAS; STEVENS, DAVID**
  - ❖ Internetworking with TCP/IP: volume 3: client-server programming and applications
  - ❖ Prentice Hall
  - ❖ 1993