

© 2004-2013 Volnys Bernal 1

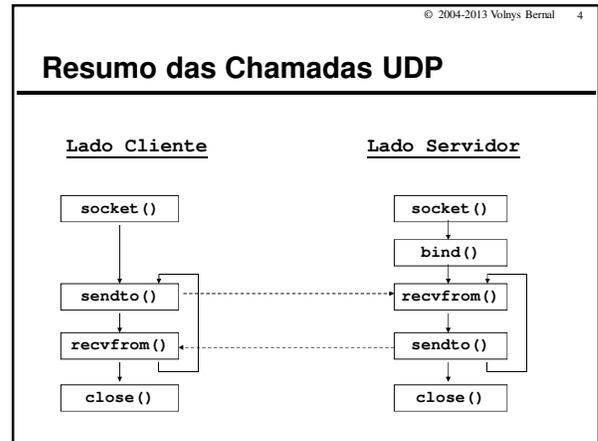
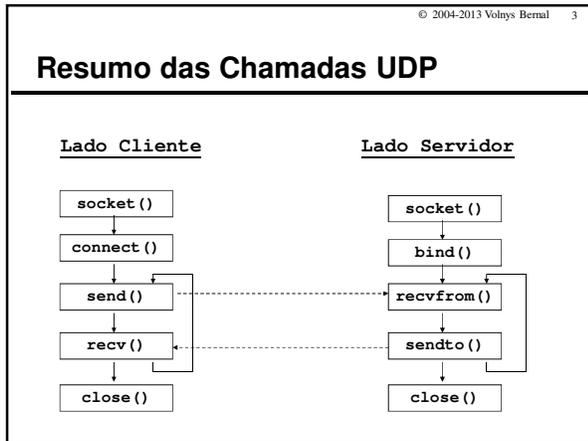
## Servidor UDP

**Volnys Borges Bernal**  
 volnys@lsi.usp.br  
<http://www.lsi.usp.br/~volnys>



© 2004-2013 Volnys Bernal 2

## Resumo das Chamadas UDP

© 2004-2013 Volnys Bernal 5

## Chamada socket()



© 2004-2013 Volnys Bernal 6

### Chamada socket()

- ❑ **Objetivo**
  - ✦ Criar um novo socket (plug de comunicação)
- ❑ **Resultado**
  - ✦ Retorna um descritor de arquivo.
- ❑ **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).
  - ✦ Endereços IP e portas são informados nas chamadas bind() (lado servidor) e connect() (lado cliente).

© 2004-2013 Volnys Bernal 7

## Chamada socket()

---

❑ **Sintaxe**

```
#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\_INET (IPv4)  
•PF\_INET6 (IPv6)  
•PF\_X25 (X25)

Tipo da comunicação:  
•SOCK\_STREAM (TCP)  
•SOCK\_DGRAM (UDP)  
•SOCK\_RAW (IP)

© 2004-2013 Volnys Bernal 8

## Chamada socket()

---

❑ **Tipo de serviço**

- ❖ SOCK\_STREAM
  - Fluxo de bytes sem delimitação
  - Full duplex
  - Orientada a conexão
  - Comunicação confiável:
    - sem perda de dados, sem duplicação, entrega na ordem
  - Deve ser utilizado com o protocolo TCP
  - Chamadas para transmissão e recepção de dados:
    - read(), write() ou send(), recv()
- ❖ SOCK\_DGRAM
  - Datagrama (mensagens)
  - Deve ser utilizado com o protocolo UDP
  - 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()

© 2004-2013 Volnys Bernal 9

## Chamada socket()

---

❑ **Exemplo de criação de socket UDP**

```
#include <sys/socket.h>
int sd; // socket descriptor
...
sd = socket(PF_INET, SOCK_DGRAM, 17);
if (sd == -1)
{
    perror("Erro na chamada socket");
    exit(1);
}
...
❑ Obs:

- ❖ O valor 17 representa o protocolo UDP, foi obtido consultando /etc/protocols. Pode também ser obtido através de resolução de nomes via getprotobyname().

```

© 2004-2013 Volnys Bernal 10

## Chamada bind()



© 2004-2013 Volnys Bernal 11

## 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

© 2004-2013 Volnys Bernal 12

## Chamada bind()

---

❑ **Chamada bind()**

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

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)

© 2004-2013 Volnys Bernal 13

### Chamada bind()

❑ Exemplo de utilização

```

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");
    
```

© 2004-2013 Volnys Bernal 14

### Chamada recvfrom()



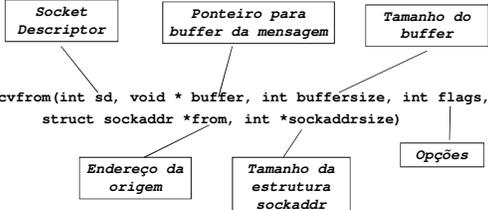
© 2004-2013 Volnys Bernal 15

### 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)
    
```



© 2004-2013 Volnys Bernal 16

### 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");

. . .
    
```

© 2004-2013 Volnys Bernal 17

### 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

© 2004-2013 Volnys Bernal 18

### Chamada sendto()



© 2004-2013 Volnys Bernal 19

### 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

Socket  
Descriptor

Ponteiro para buffer  
da mensagem

Tamanho da  
mensagem

```
int sendto(int sd, void *txbuffer, int msgsize,
           int flags, struct sockaddr *to, int sockaddrsz)
```

Opções

Endereço do  
destino

Tamanho da  
estrutura  
sockaddr

© 2004-2013 Volnys Bernal 20

### 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");
...
```

© 2004-2013 Volnys Bernal 21

## Exercício



© 2004-2013 Volnys Bernal 22

### Exercício

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

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

© 2004-2013 Volnys Bernal 23

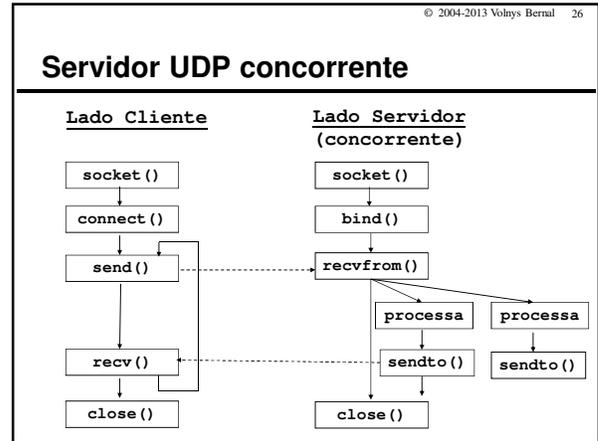
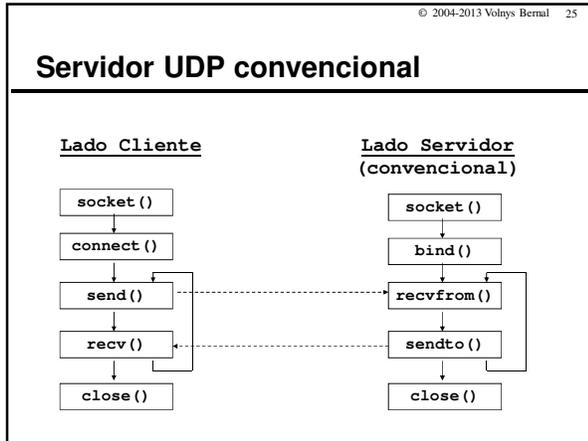
## Servidor Não Concorrente x Concorrente



© 2004-2013 Volnys Bernal 24

### Servidor não concorrente x concorrente

- ❑ Servidor não concorrente
  - ❖ Processa uma requisição por vez
- ❑ Servidor concorrente
  - ❖ Tem capacidade para processar mais que uma requisição simultaneamente
  - ❖ Mais complexo
  - ❖ Mais difícil de implementar
  - ❖ Necessita uma implementação multithreaded



© 2004-2013 Volnys Bernal 27

## Exercício

© 2004-2013 Volnys Bernal 28

### 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);
}
        
```

© 2004-2013 Volnys Bernal 29

## Referências Bibliográficas

© 2004-2013 Volnys Bernal 30

### Referências Bibliográficas

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