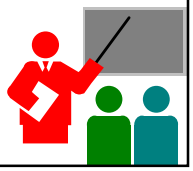
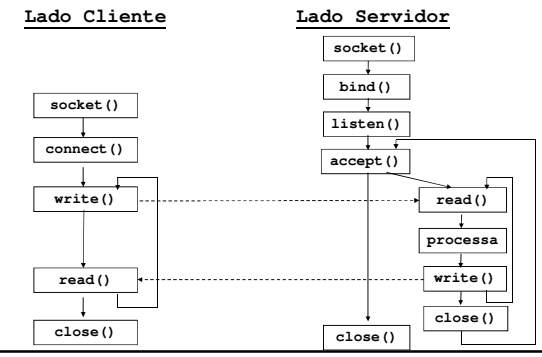
Servidor TCP

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

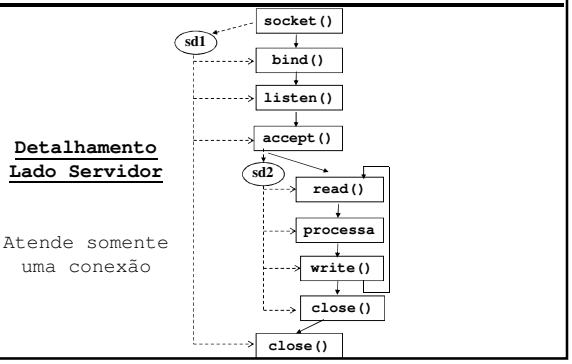
Resumo das Chamadas TCP



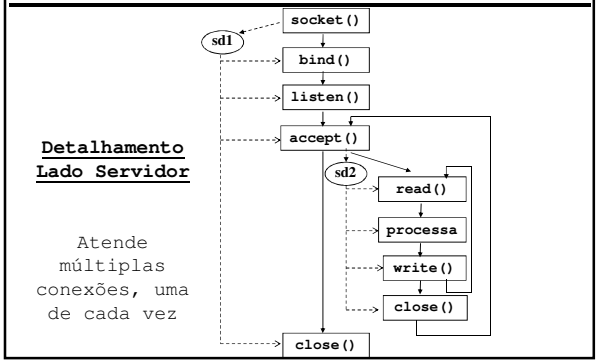
Resumo das Chamadas TCP



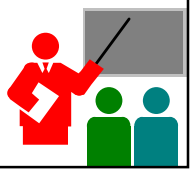
Resumo das Chamadas TCP



Resumo das Chamadas TCP



Chamada socket()



Chamada socket()

- **Objetivo**
  - Criar um novo socket (plug de comunicação)
- **Resultado**
  - Retorna um descritor de arquivo.
- **Sintaxe**

```
int socket (int domain, int type, int protocol)
```

- **Observação:**
  - Quando um socket é criado ele não possui nenhuma informação armazenada (endereços IPs e portas).
  - Endereços IPs e portas são informados nas chamadas bind() (lado servidor) e connect() (lado cliente).

Chamada socket()

- **Sintaxe**

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

Socket descriptor

Pilha de protocolos
- PF_INET
- PF_INET6
- PF_X25

Tipo da comunicação
- SOCK_STREAM (TCP)
- SOCK_DGRAM (UDP, TCP)
- SOCK_RAW (IP)

Id. do protocolo
- UDP (17)
- TCP (6)

Chamada socket()

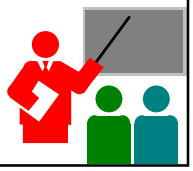
- **Exemplo de criação de socket TCP**

```
#include <sys/socket.h>
int sd;

. . .
sd = socket(PF_INET, SOCK_STREAM, 6)
if (sd == -1)
    perror("Erro na chamada socket");
. . .
```

- **Obs:**
  - O valor 6 representa o protocolo TCP, foi obtido consultando /etc/protocols. Pode também ser obtido através de resolução de nomes via getprotobyname().

Chamada bind()



Chamada bind()

- **Objetivo**
  - Associar um *socket address* (IP+porta) ao socket
  - Deve ser utilizado no lado servidor
- **Valor retornado pela função**
  - 0: sucesso
  - –1: erro

Chamada bind()

- **Sintaxe**

Descritor do socket

Endereço IP da interface local. Pode ser o endereço IP de:
- *Uma interface específica*
- *Todas interfaces locais (INADDR_ANY)*

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

Resultado da chamada: sucesso ou erro

Tamanho da estrutura de endereço (sockaddr)

 13

Chamada bind()

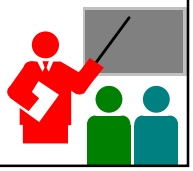
- **Exemplo de utilização**

```
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(socketdescriptor,
              (struct sockaddr *) &mylocal_addr,
              sizeof(struct sockaddr_in));
if (status == -1)
    perror("Erro na chamada bind");
```

 14

Chamada listen()



 15

Chamada listen()

- **Objetivo**
  - Abrir a porta na qual o servidor ira aguardar conexões TCP
  - As conexões são aceitas através da ativação de accept()
- **Valor retornado pela função**
  - 0: sucesso
  - -1: erro

 16

Chamada listen()

- **Sintaxe**

Resultado da função:
0: sucesso
-1: erro

Descritor do socket

```
int listen(int sd,
           int queuelenght)
```

Qde máxima de conexões pendentes sem aceite

 17

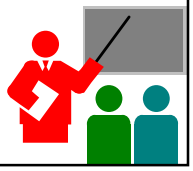
Chamada listen()

- **Exemplo:**

```
#define QLEN 10
...
status = listen(sd,QLEN);
if (status != 0)
        {
        perror("Erro na chamada listen()");
        exit(1);
        }
```

 18

Chamada accept()



 19

Chamada accept()

- **Objetivo**
  - Aceitar uma nova conexão TCP.
  - Accept() extrai a primeira conexão da fila e gera um novo *socket descriptor* para esta conexão.
  - O socket original não é afetado por esta chamada.
- **Valor retornado pela função**
  - Valor não negativo: sucesso, sendo este o valor do novo socket descriptor
  - -1: erro

 20

Chamada accept()

- **Sintaxe**

Valor retornado:
Positivo: sucesso, correspondendo ao valor do novo socket
-1: erro

Descritor do socket

```
int accept (int sd,
            struct sockaddr *addr,
            socklen_t *addrlen)
```

Tamanho da estrutura socketaddr

Ponteiro para uma estrutura sockaddr que conterá o endereço (socket address) do parceiro de comunicação

 21

Chamada accept()

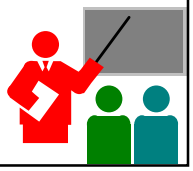
- **Exemplo**

```
int                 newsd;
int                 size;
struct sockaddr_in clientaddr;

....
size = sizeof(clientaddr);
newsd = accept( sd,
                (struct sockaddr *) &clientaddr,
                (socklen_t *)       &size);
if (newsd < 0)
        {
        perror("Erro na chamada accept()");
        exit(1);
        }
....
```

 22

Chamada read()



 23

Chamada read()

- **Objetivo**
  - Recepção/leitura de dados de um descritor
    - Descritor: descritor sockets, descritor de arquivo, ...
  - Pode ser utilizada por cliente ou servidor
- **Valor retornado pela função**
  - >0: quantidade de bytes lidos
  - 0: end of file
  - -1: erro

 24

Chamada read()

- **Sintaxe:**

```
#include <unistd.h>

int read(int sd, void *buf, int buffersize)
```

Socket Descriptor

Tamanho do buffer

Ponteiro para o buffer (end. do buffer de recepção)

 25

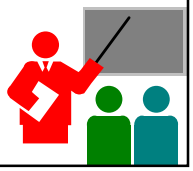
Chamada read()

- **Exemplo:**

```
. . .
status = read(sd, bufferp, buffersize)
if (status == -1)
    perror("Erro na chamada read");
. . .
```

 26

Chamada write()



 27

Chamada write()

- **Objetivo**
  - Transmissão/escrita de dados em um descritor
    - Descritor: descritor sockets, descritor de arquivo, ...
  - Pode ser utilizada por cliente ou servidor
- **Valor retornado pela função**
  - Positivo: quantidade de bytes escritos
  - -1: erro

 28

Chamada write()

- **Sintaxe**

```
#include <unistd.h>

int write(int sd, void *buf, int count)
```

Socket Descriptor

Tamanho da mensagem

Ponteiro para mensagem (end. do buffer da mensagem)

 29

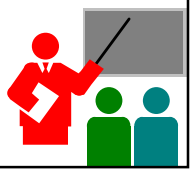
Chamada write()

- **Exemplo:**

```
. . .
status = write(sd,txbuffer, strlen(txbuffer)+1)
if (status == -1)
  perror("Erro na chamada write");
. . .
```

 30

Chamada close()



 31

Chamada close()

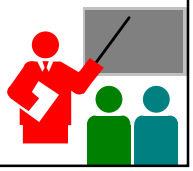
- Permite fechar o socket (assim como é feito com arquivo)
- Existem dois sockets abertos. É importante fechar os dois sockets.
- Código de exemplo para fechar um socket:

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

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

 32

Exercício



 33

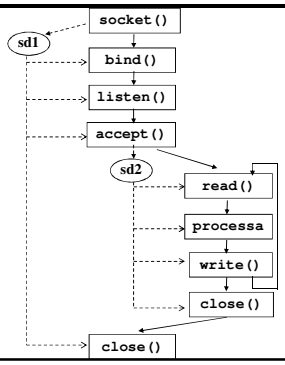
Exercício

1. **Implemente um servidor TCP echo que transforma para maiúsculas.**
   - O programa deve atender a um cliente TCP e, quando receber a mensagem "quit" deve encerrar a conexão com o cliente e terminar o programa
   - Dicas:
     - Utilize a rotina de biblioteca `toupper()` para converter um caractere minúsculo para maiúsculo.
     - Utilize a rotina de biblioteca `strcmp()` para comparar a string recebida com "quit".

```
#include <string.h>
If (strcmp(bufferp,"quit")==0)
    /* igual */
```

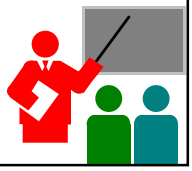
 34

Exercício



 35

Servidor Não Concorrente x Concorrente

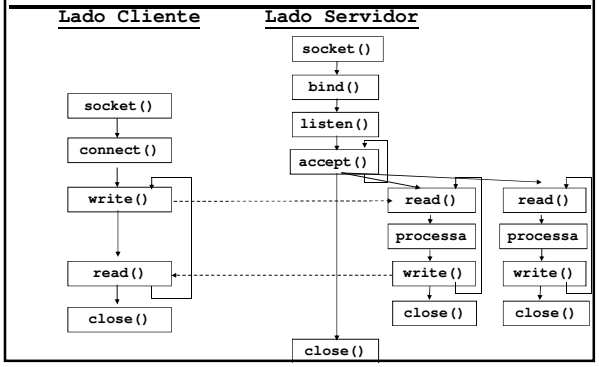


 36

Servidor não concorrente x concorrente

- **Não concorrente**
  - Processa uma requisição por vez
- **Concorrente**
  - Tem capacidade para processar mais que uma requisição simultaneamente
  - Mais complexo
  - Mais difícil de implementar
  - Necessita uma implementação multithreaded

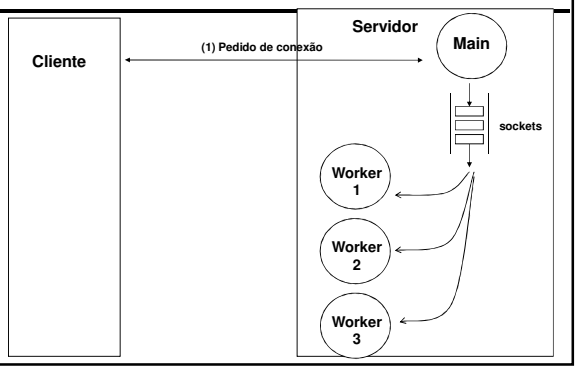
Servidor TCP concorrente



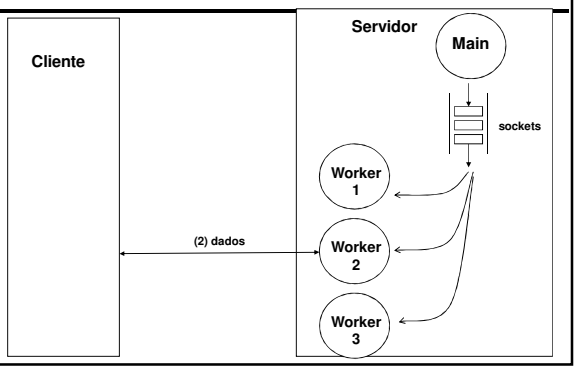
Exercício

- **Fazer um servidor TCP echo concorrente.**
  - Dica: existem diversas formas de implementar o controle dos threads:
  - (a) Utilizar uma tabela de controle de conexões estabelecidas com cada entrada da tabela contendo os seguintes campos: ocupado, estrutura dos threads (thread_t), semáforo para o thread e socket (new socket).
  - (b) Utilizar a estrutura produtor-consumidor. O produtor é o thread principal: produz novas conexões (socket descriptors). O consumidor são os threads de trabalho: consomem conexões (socket descriptors). Um consumidor, ao receber o socket descriptor fica responsável pela interação com o cliente até que a conexão seja encerrada.

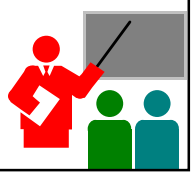
Exercício



Exercício



Referências Bibliográficas



Referências Bibliográficas

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