

IPC: introdução a “sockets”

Comunicação entre dois Processos em Execução
Os processos podem estar em computadores distintos

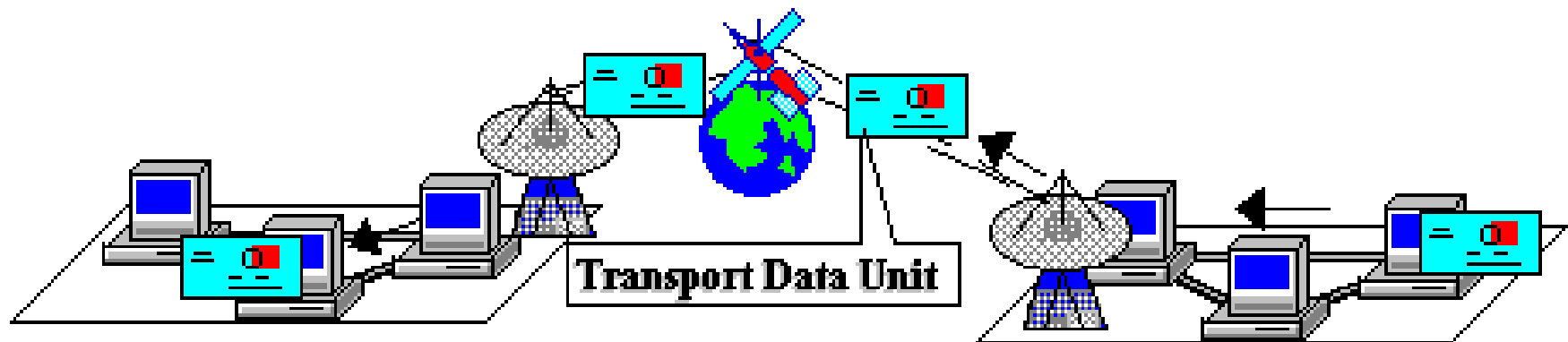


Fig. 1: Unidade de transporte de dados do TCP/IP



Sumário

- Introdução
 - Internet
- Networks – Os Básicos
 - Vista geral
 - TCP
 - UDP
- Portas e Sockets
- Programação cliente-servidor através de sockets
- Exemplo



Introdução

- A Internet e WWW tem evoluído para um meio de comunicação global e ubíqua.
 - Mudou a maneira de fazer ciência, comércio e engenharia.
 - Mudou inclusivamente as relações humanas
 - Influenciou a maneira como aprendemos, conduzimos as nossas relações pessoais (chats, email, instante messenger, redes sociais etc.), quase todos os aspetos da vida humana moderna.

Protocolos da Internet

- Na Internet, a comunicação entre máquinas de plataformas e sistemas operativos distintos faz-se através de duas classes de protocolos da Internet:
 - **TCP/IP** (Transmission Control Protocol/Internet Protocol) que especifica a *data transport layer* de comunicação.
 - Cada transacção de dados entre dois processo em dois computadores é um fluxo de bytes em parcelas designadas por *transport data units*;
 - Application Protocolos : que incluem FTP, HTTP e NEWS.
 - Esta camada de protocolos é responsável pela estrutura interna e a semântica das *data units* em cada transacção.

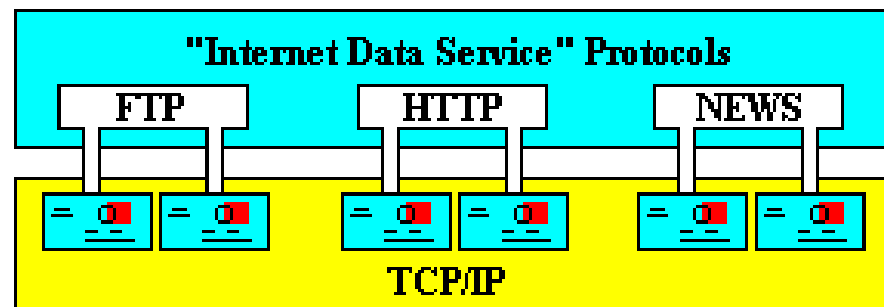
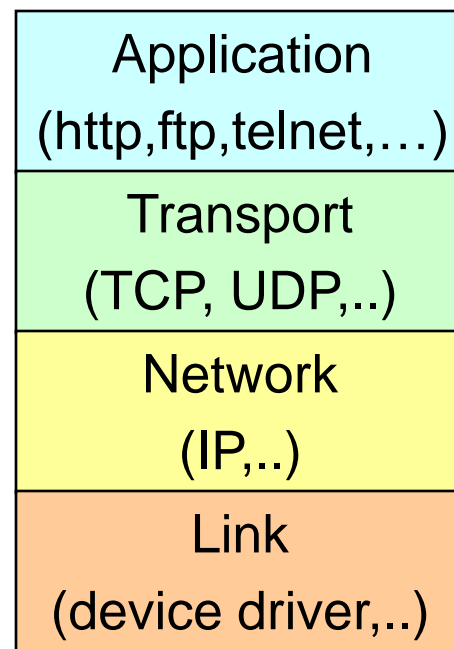


Fig.1: Protocolos Internet

Networking Basicos

- TCP (Transport Control Protocol)
- RFC 793 September 1981
<https://www.ietf.org/rfc/rfc793.txt>
- Protocolo de transporte orientado por conexão que fornece um fluxo de dados entre processos.
 - A entrega de dados correta, sem erros, no ordem certa é **garantida**.
- Exemplos de aplicações:
 - Browsers (HTTP)
 - File Transfer FTP
 - Remote Connection (Telnet)

Fig 3: Pilha TCP/IP

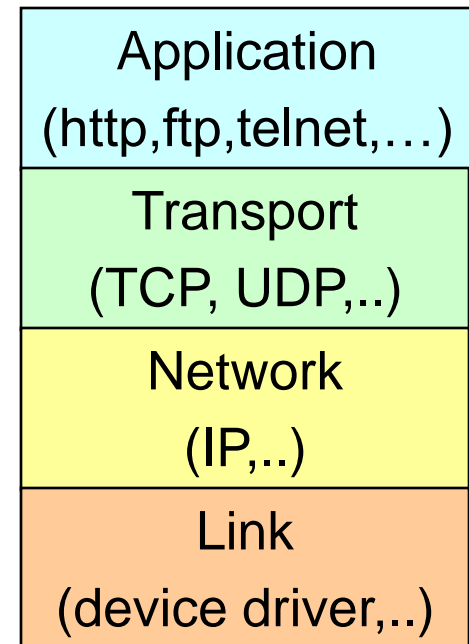


Ref: <http://pt.wikipedia.org/wiki/TCP>

Networking Basicos

- UDP (User Datagram Protocol)
- Protocolo que trate do envio independente de pacotes de dados, chamados *datagrams*, dum processo para um outro
 - sem nenhuma garantia sobre a chegada ou não destes dados.
- Exemplos de aplicações:
 - Clock server
 - Ping

Fig. 4: Pilha TCP/IP



Ref: http://pt.wikipedia.org/wiki/Protocolo_UDP

Protocolo TCP/IP:

Tradução de nomes

- Um domínio tem sempre uma máquina **DNS** (Domain Name Server) que mantém uma tabela de correspondências entre os nomes e os números IP.
- Ao invés dos números IP, os nomes não são estritamente necessários à comunicação entre computadores.

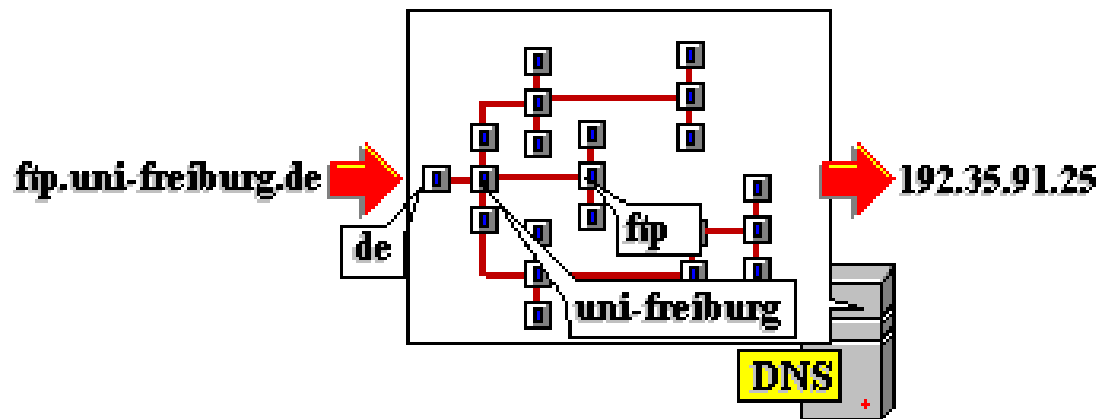


Fig.9: Tradução de nomes

Comunicação cliente-servidor: exemplo do protocolo HTTP

- Na Internet (http), a comunicação cliente-servidor tem lugar sobre uma ligação TCP/IP.
- Os pedidos e as respostas são mapeados sobre *transport data units* de TCP/IP.

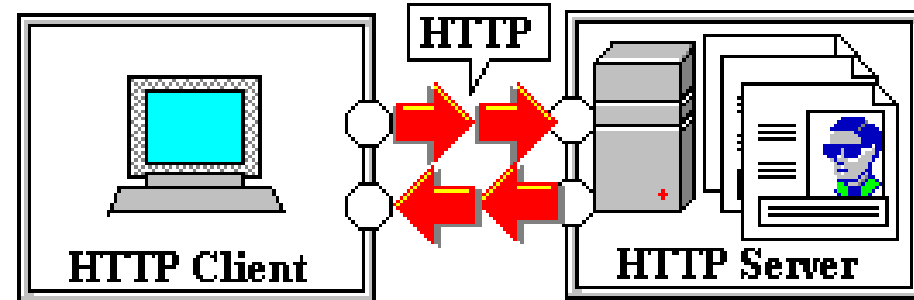


Fig.8: HTTP: um protocolo cliente-servidor

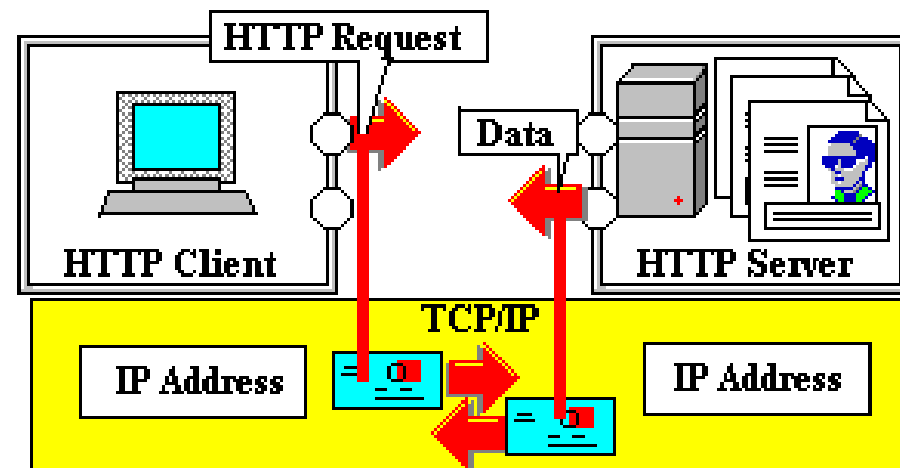


Fig.10: HTTP sobre TCP/IP

Portas

- Os protocolos TCP (e UDP) utilizam *ports* para mapear os dados a chegar para um processo em particular.

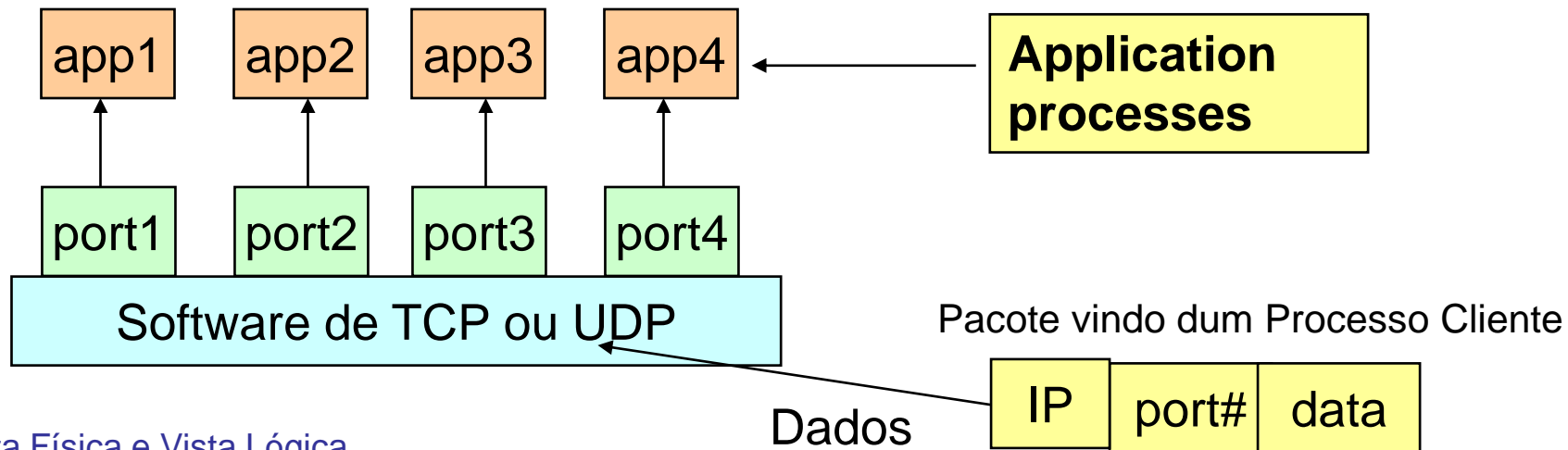
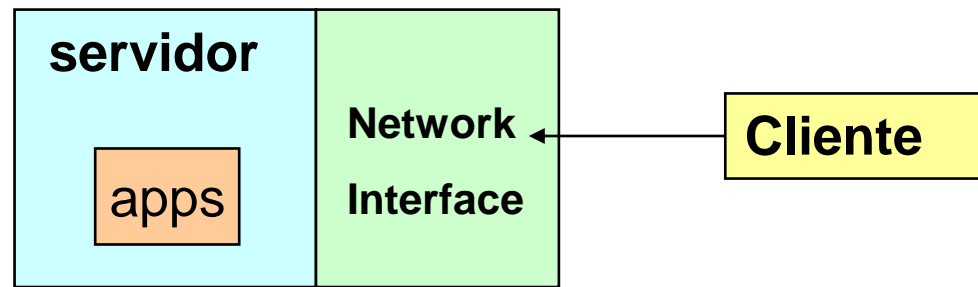


Fig: Vista Física e Vista Lógica



Portas

- **Port** um valor positivo (16-bit integer value)
- Algumas portas estão reservados para serviços conhecidos:
 - ftp 21/tcp
 - telnet 23/tcp
 - ssh 22/tcp
 - smtp 25/tcp
 - login 513/tcp
- Processos/Serviços dos utilizadores normalmente utilizam portas com valores ≥ 1024

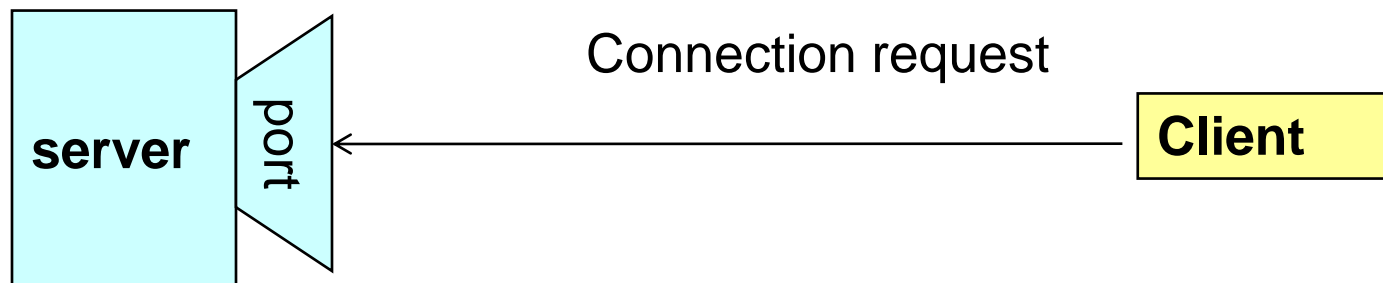


Sockets

- Sockets fornecem um interface para a programação (API)
- Um socket (“ficha”) é um ponto terminal numa ligação de comunicação entre duas entidades numa rede.
- A analogia pode ser feita entre o sistema telefónico ou eléctrica
 - o aparelho ligue-se a uma ficha para comunicar. Como podemos ter apenas uma ligação física este será dividido logicamente em vários “portas”
- A programação usando Sockets é feito com as operações de I/O de baixo nível de ficheiros
 - Um socket, visto dum programa (Linux), não é mais do que um ficheiro aberto tratado de mesma maneira como o resto de sistema I/O – através dum descritor de ficheiro e read() e write().
- Comunicação baseado em Sockets é independente da linguagem de programação.
 - Quer dizer que um programa que utiliza o modelo de Sockets escrito em C pode comunicar com outro escrito em Java ou outra linguagem qualquer...

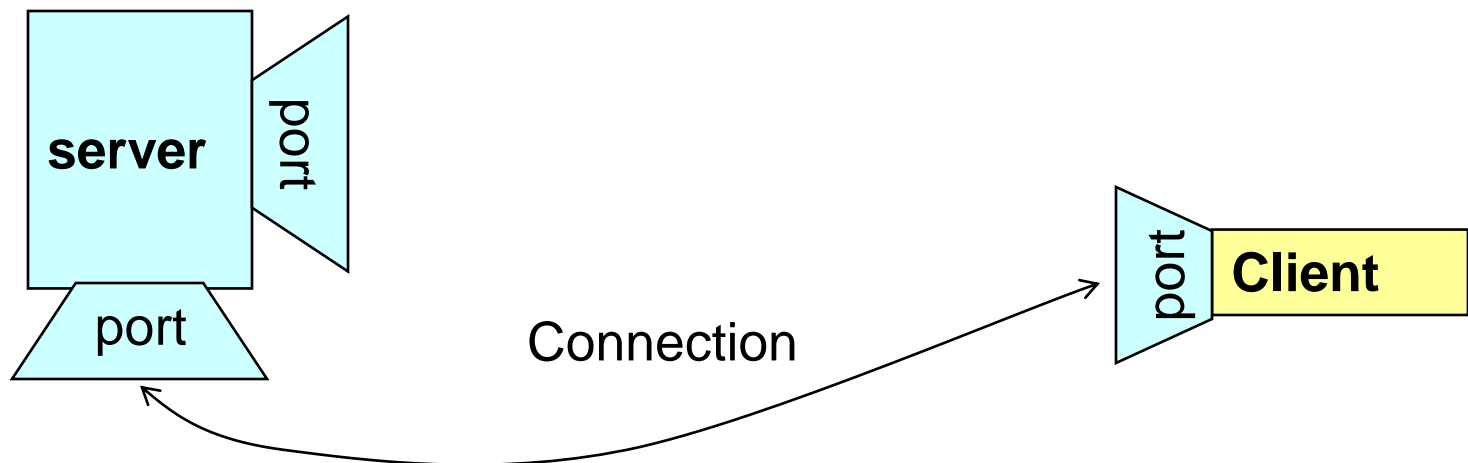
Programação cliente-servidor através de **sockets**

- Um servidor (processo) a executar num dado computador tem um "socket" ligado ("**bound**") a uma **porta** específica.
- O servidor espera, a **escutar** ("listen") no socket até que um cliente faz um pedido.



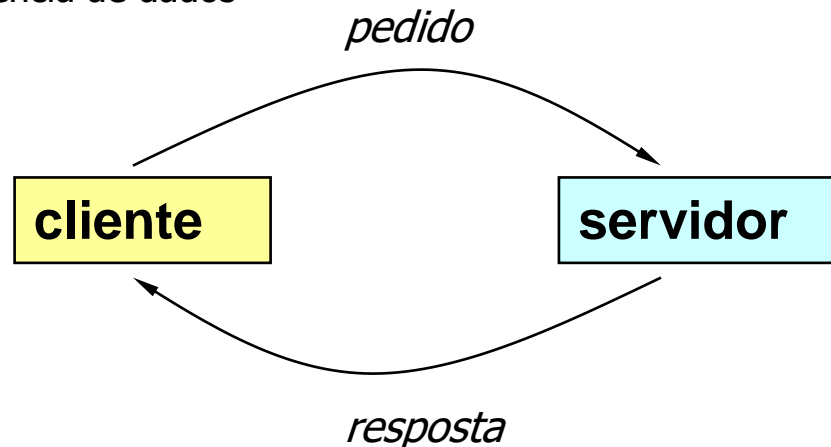
Programação cliente-servidor através de **sockets**

- Na altura de aceitar uma ligação o servidor obtêm um novo socket para que possa continuar a escutar no socket original



Programação cliente-servidor através de **sockets**

- A **comunicação** cliente-servidor requer a definição de 5 itens:
 - o endereço IP do cliente;
 - o porto do cliente;
 - o endereço IP do servidor;
 - o porto do servidor.
 - o protocolo a usar;
 - Uma sequencia de pedidos e respostas que é uma transmissão e recepção duma sequencia de dados



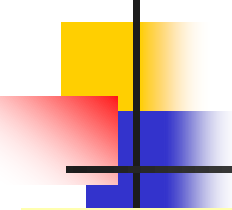


Cliente – Servidor Exemplo

Exemplo Simplificado

sem verificação de erros

sem ciclo no servidor



Comunicação com sockets com conexão (stream sockets)

- Os passos básicos do **servidor** são :

- Cria socket;
- Associa endereço a socket (OBRIGATÓRIO);
- Aceita conexão;
- Comunica com o cliente (read and write);
- Fecha socket.

- Os passos básicos do **cliente** são :

- Cria socket;
- Associa endereço a socket (OPCIONAL);
- Estabelece conexão;
- Comunica com o servidor (read and write);
- Fecha conexão.



Exemplo 1

Apresente-se o código sem verificação de erros nas chamadas ao sistema

O ficheiro

server.c

cliente.c

example.h



Exemplo

```
//example.h
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include <netdb.h>
```



Servidor 1: exemplo

```
/* server.c - a stream socket server demo */
#include "example.h"

#define MYPORT 3490 // the port users will be connecting to
#define BACKLOG 10 // how many pending connections queue will hold

int main(void)
{
    int sockfd;                // listen on sock_fd
    int new_fd;                // new connection on new_fd
    struct sockaddr_in my_addr; // my address information
    struct sockaddr_in their_addr; // connector's address information
    socklen_t sin_size;        // tamanho dos sockets
    int yes=1;

    /* Criar um Socket com propriedades */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));

    /* Construção do endereço do servidor */
    my_addr.sin_family = AF_INET; // host byte order
    my_addr.sin_port = htons(MYPORT); // short, network byte order
    my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
    memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct
}
```



Servidor 1 : exemplo (cont.)

```
/* LIGAR Socket com Endereco */
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));

/* Começar a escutar .. */
listen(sockfd, BACKLOG);

/*A Nova ligação sera aceita */

sin_size = sizeof(struct sockaddr_in);

new_fd=accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);

printf("server: got connection from %s\n", inet_ntoa(their_addr.sin_addr));

/*Escreve qualquer coisa - 14 bytes -para o cliente*/
send(new_fd, "Hello, world!\n", 14, 0);

return 0;
}
```



Cliente: exemplo

```
/*
** client.c - a stream socket client demo
*/
#include "example.h"

#define PORT 3490          // the port client will be connecting to
#define MAXDATASIZE 100   // max number of bytes we can get at once

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct hostent *he;
    struct sockaddr_in their_addr; // connector's address information

    if (argc != 2) {
        fprintf(stderr, "usage: client hostname\n");
        exit(1);
    }
}
```



Cliente: exemplo (cont.)

```
he=gethostbyname(argv[1]) // get the host info
if (null==he){
    perror("Host Info Failed");
    exit(1);
}

sockfd = socket(AF_INET, SOCK_STREAM, 0);

their_addr.sin_family = AF_INET; // host byte order
their_addr.sin_port = htons(PORT); // short, network byte order
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(&(their_addr.sin_zero), '\0', 8); // zero the rest of the struct

connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr) );

numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0);

buf[numbytes] = '\0';
printf("Received from server: %s",buf);
close(sockfd);

return 0;
}
```



Funcionamento

```
./server
```

```
server: got connection from 193.136.66.4
```

```
./client neve.di.ubi.pt
```

```
Received from server: Hello, world!
```

```
ubuntu >./server &
```

```
[1] 280
```

```
ubuntu >./client localhost
```

```
server: got connection from 127.0.0.1
```

```
Received from server: Hello, world!
```

```
[1]+  Done                  ./server
```



Exemplo 2

- Neste exemplo o servidor não vai morrer.
- Cada pedido dum cliente será servido num novo processo
- Apresentamos apenas as alterações necessários ao servidor
 - `server2.c`
- O programa client não é alterado



Servidor 2 : exemplo (cont.)

```
/* aceitar novas ligações e servi-las */
while(1) { // main accept() loop

    sin_size = sizeof(struct sockaddr_in);

    new_fd=accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);

    printf("server: got connection from %s\n", inet_ntoa(their_addr.sin_addr));

    if ( fork()==0 ) {        // this is the child process
        close(sockfd);        // child doesn.t need the listener so close it
        send(new_fd, "Hello, world!\n", 14, 0);
        close(new_fd);
        exit(0);              //terminar filho
    }
    close(new_fd); // parent doesn.t need this
}

return 0;
}
```



Funcionamento 2

```
[crocker@neve cli-ser]$ ./server &  
[1] 22236  
[crocker@neve cli-ser]$ server: got connection from 193.136.66.4  
server: got connection from 193.136.66.4  
server: got connection from 193.136.66.4
```

```
[crocker@neve cli-ser]$ ./client neve.di.ubi.pt  
Received: Hello, world!  
[crocker@neve cli-ser]$ ./client neve.di.ubi.pt  
Received: Hello, world!  
[crocker@neve cli-ser]$ ./client neve.di.ubi.pt  
Received: Hello, world!  
[crocker@neve cli-ser]$
```



Testando servidores com Telnet

- O programa `telnet` é util para testar servidores que transmitam **ASCII** strings sobre ligações Internet
 - Web servers
 - Mail servers
 - etc.
- Utilização:
 - `telnet <host> <portnumber>`
 - Criar uma ligação com um servidor a executar no servidor `<host>` a escutar na porta `<portnumber>`.



Funcionamento 3

```
[crocker@neve cli-ser]$ telnet neve.di.ubi.pt 3490
```

```
Received from server: Hello, world!
```

```
[crocker@neve cli-ser]$ telnet neve.di.ubi.pt 3490
```

```
Received from server: Hello, world!
```

```
[crocker@neve cli-ser]$
```



Exemplo 3 Signal Handling

Reaping the Child Processes

Neste exemplo vamos inserir um pequeno **tratamento de sinais** no nosso servidor. Este tem o propósito de eliminar processos “zombies” criados pelo servidor.

Ao fazer `fork()` o servidor vai criar filhos que a terminar enviem um sinal ao processo progenitor.

Os filhos ficam num estado “Terminado” mas não morrem por completo (e portanto o SO não pode limpar totalmente todos os recursos consumados pelo processo) **até** o processo progenito apanhar e tratar do sinal (por exemplo via `wait`)



Server 2 e os seus Zombies

```
[user@f12vboxpaulc examples]$ ./server2 &
```

```
server: got connection from 127.0.0.1
```

```
server: got connection from 127.0.0.1
```

```
server: got connection from 127.0.0.1
```

```
[user@f12vboxpaulc examples]$ ps
```

PID	TTY	TIME	CMD
2557	pts/0	00:00:00	bash
2621	pts/0	00:00:00	server2
2654	pts/0	00:00:00	server2 <defunct>
2664	pts/0	00:00:00	server2 <defunct>
2666	pts/0	00:00:00	server2 <defunct>



Zombies

ubuntu >ps

PID	TTY	TIME	CMD
17	tty1	00:00:00	bash
299	tty1	00:00:00	server2
301	tty1	00:00:00	server2 <defunct>
303	tty1	00:00:00	server2 <defunct>
305	tty1	00:00:00	server2 <defunct>
307	tty1	00:00:00	server2 <defunct>
325	tty1	00:00:00	ps

ubuntu >**killall** server2

[1]+ Terminated ./server2

ubuntu >ps

PID	TTY	TIME	CMD
17	tty1	00:00:00	bash
327	tty1	00:00:00	ps

ubuntu >



Servidor 3: exemplo

```
/* server3.c - a stream socket server demo */

#define MYPORT 3490 // the port users will be connecting to
#define BACKLOG 10 // how many pending connections queue will hold

void sigchld_handler(int s) {
    while(wait(NULL) > 0);
}

int main(void)
{
    int sockfd, new_fd; // listen on sock_fd, new connection on new_fd
    struct sockaddr_in my_addr; // my address information
    struct sockaddr_in their_addr; // connector's address information
    int sin_size;
    struct sigaction sa; //signal handling
    int yes=1;
```




Servidor 3: exemplo (cont.)

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

setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));

my_addr.sin_family = AF_INET;          // host byte order
my_addr.sin_port = htons(MYPORT);      // short, network byte order
my_addr.sin_addr.s_addr = INADDR_ANY;  // automatically fill with my IP
memset(&(my_addr.sin_zero), '\0', 8);  // zero the rest of the struct

bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));

listen(sockfd, BACKLOG);

//SETUP SIGNAL HANDLING dentro da estrutura "sa" e activar com sigaction

sa.sa_handler = sigchld_handler; // reap all dead processes
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
sigaction(SIGCHLD, &sa, NULL);
```



Servidor 3 : exemplo (cont.)

```
/* aceitar novas ligações e servi-las */
```

```
while(1) { // main accept() loop

    sin_size = sizeof(struct sockaddr_in);

    new_fd=accept(sockfd,(struct sockaddr *)&their_addr,&sin_size);

    printf("server: got connection from %s\n", inet_ntoa(their_addr.sin_addr));

    if ( fork()==0 ) {        // this is the child process
        close(sockfd);        // child doesn.t need the listener so close it
        send(new_fd, "Hello, world!\n", 14, 0);
        close(new_fd);
        exit(0);              //terminar
    }
    close(new_fd); // parent doesn.t need this
}

return 0;
}
```



Referências

- *BSD Sockets Quick and Dirty* Jim Frost
- *Unix Network Programming* by W. Richard Stevens
- *Computer Systems* Bryant and O'hallaron (Capítulo 12)