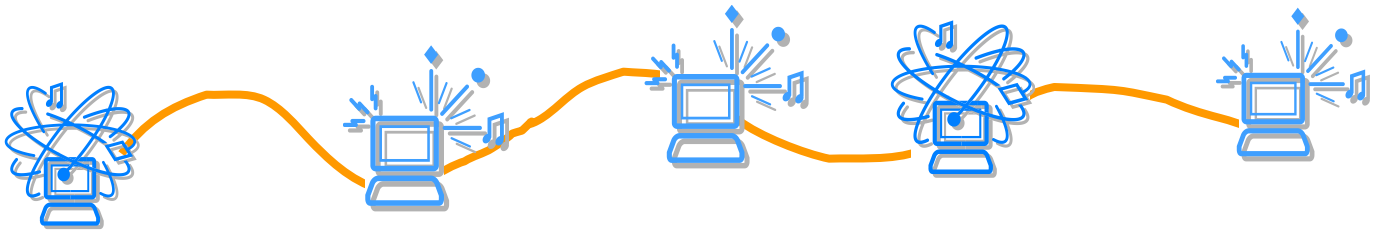
Programação TCP/IP (sockets)

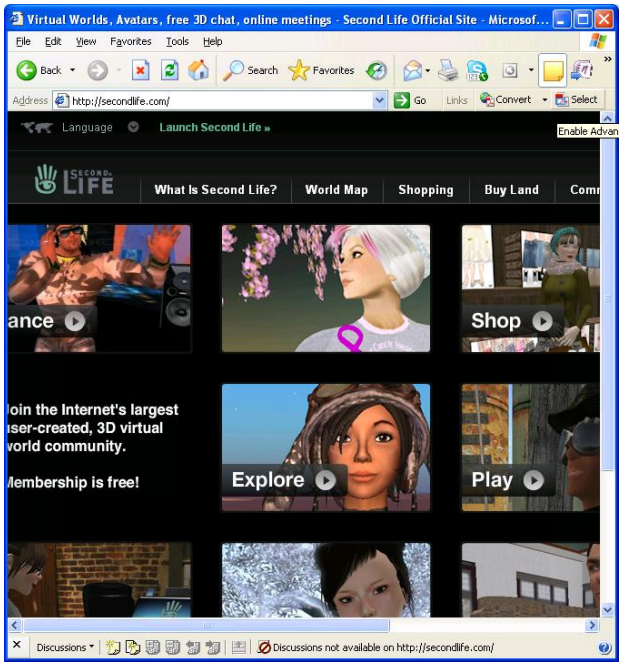


Instituto Superior de Engenharia de Lisboa

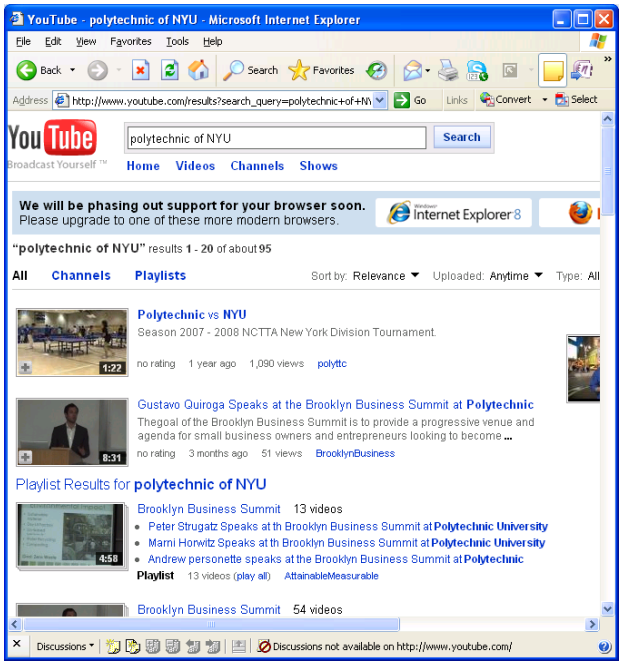
Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Redes de Computadores

Aplicações

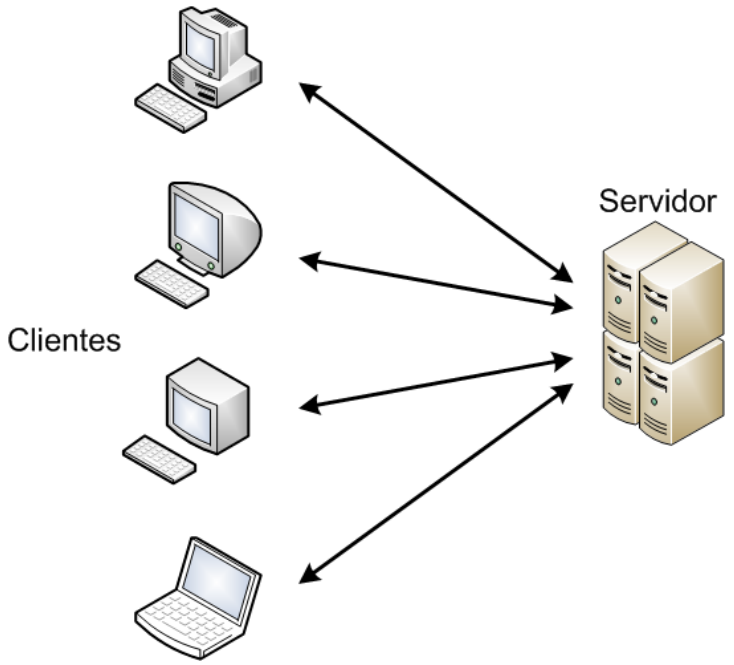

Virtual Worlds, Avatars, free 3D chat, online meetings - Second Life Official Site - Microsof...
Launch Second Life »
What Is Second Life?
World Map
Shopping
Buy Land
Shop
Explore
Play
Discussions not available on http://secondlife.com/


Facebook | Keith Ross - Microsoft Internet Explorer
facebook
Home
Profile
Friends
Inbox
Keith Ross
Wall
Info
Photos
Boxes
Video
What's on your mind?
Keith Ross Katie playing Blackbird at the 2009 LFNY recital.
KatieGuitar2009
Information
Discussions not available on http://www.facebook.com/


YouTube - polytechnic of NYU - Microsoft Internet Explorer
YouTube
polytechnic of NYU
Search
Home
Videos
Channels
Shows
We will be phasing out support for your browser soon.
Please upgrade to one of these more modern browsers.
"polytechnic of NYU" results 1 - 20 of about 95
Polytechnic vs NYU
Gustavo Quiroga Speaks at the Brooklyn Business Summit at Polytechnic
Playlist Results for polytechnic of NYU
Brooklyn Business Summit 13 videos
Brooklyn Business Summit 54 videos
Discussions not available on http://www.youtube.com/

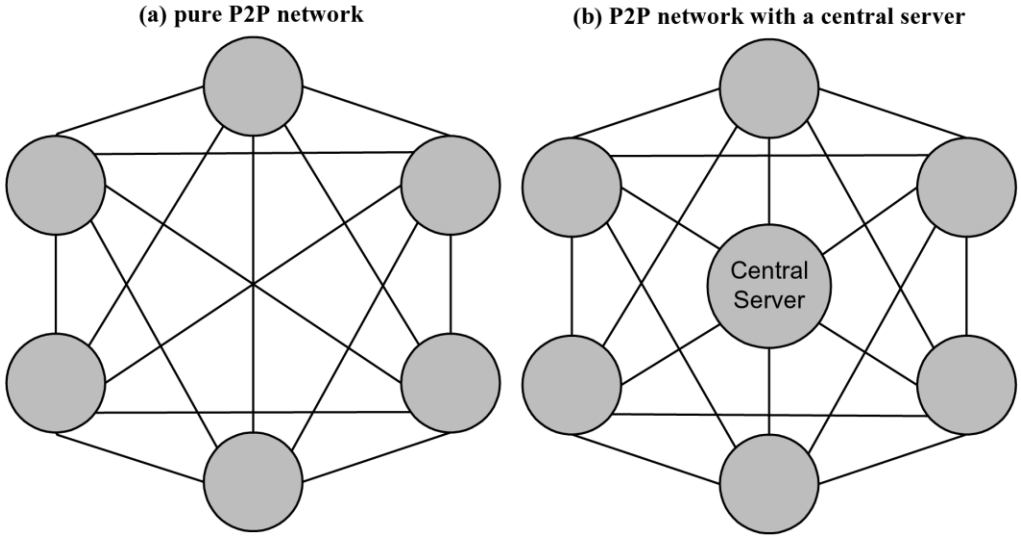
Arquitectura cliente-servidor

- Servidor:
  - Sempre ligado
  - Endereço IP permanente
  - Escalabilidade á custa de clusters de servidores
- Clientes:
  - Comunicam com o servidor
  - Efectuam ligações quando desejarem
    - Não se mantêm permanentemente ligados
  - Podem ter endereços IP dinâmicos
  - Não comunicam directamente entre si



Arquitecturas P2P

- Sem servidor central
- Os sistemas (*peers*) finais comunicam directamente
- Os sistemas estão temporariamente ligados e trocam de endereço IP
- Altamente escalável e resiliente mas difícil de gerir



Sistemas hibrídos P2P – Cliente/Servidor

- Skype:
  - Aplicação P2P de Voz sobre IP
  - Servidor central: encontrar o endereço do telefone remoto
  - Ligação de voz entre clientes: directo (não passa pelo servidor)

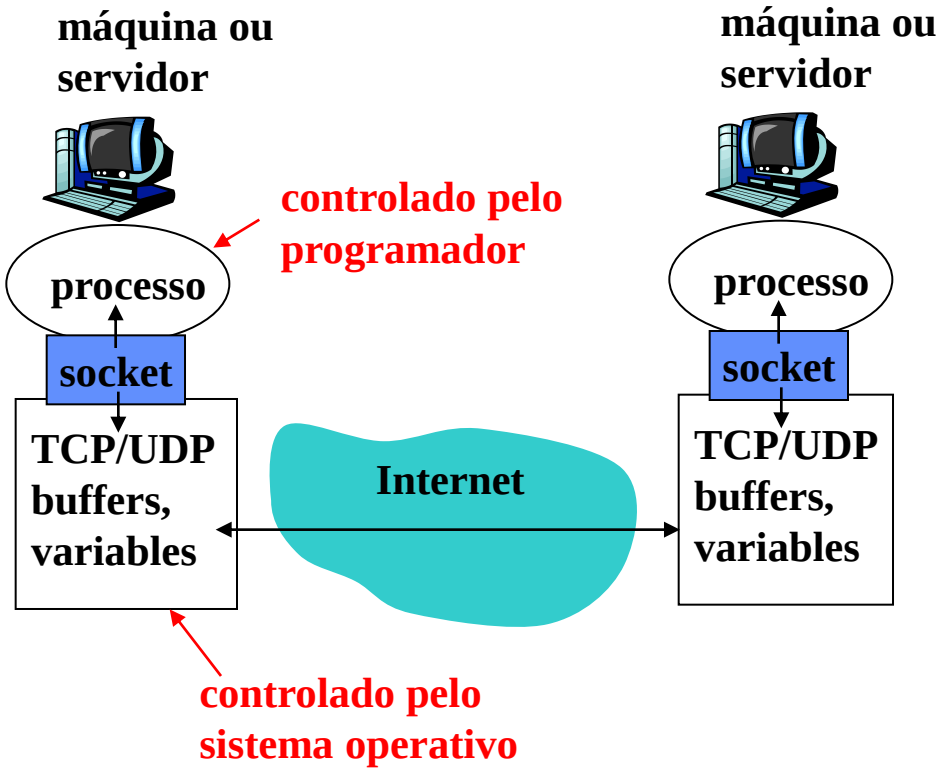
- Instant messaging:
  - Servidor central: detecção/localização dos clientes
    - Os clientes registam o seu endereço IP com o servidor central
    - Cliente contacta o servidor para conhecer o endereço IP dos contactos
  - A troca de texto entre os dois clientes é directa

Comunicação entre processos

- Processo: programa a ser executado numa máquina
  - Dentro da mesma máquina: dois processos comunicam utilizando mecanismos de IPC (inter-process communication) definidos pelo sistema operativo
  - Processos em diferentes máquinas comunicam através de mensagens
- Processo cliente: processo que inicia a comunicação
- Processo servidor: processo que espera ser contactado
- Nota: aplicações P2P possuem processos clientes/servidores

Sockets

- Interface entre a camada de aplicação e a camada de transporte
- Socket corresponde a uma API
  - Um processo recorre ao socket para enviar mensagens
  - Ao enviar uma mensagem o processo recorre à infra-estrutura de transporte TCP/UDP
  - O socket da máquina de destino é utilizado para receber a mensagem



Protocolos de transporte: serviços oferecidos

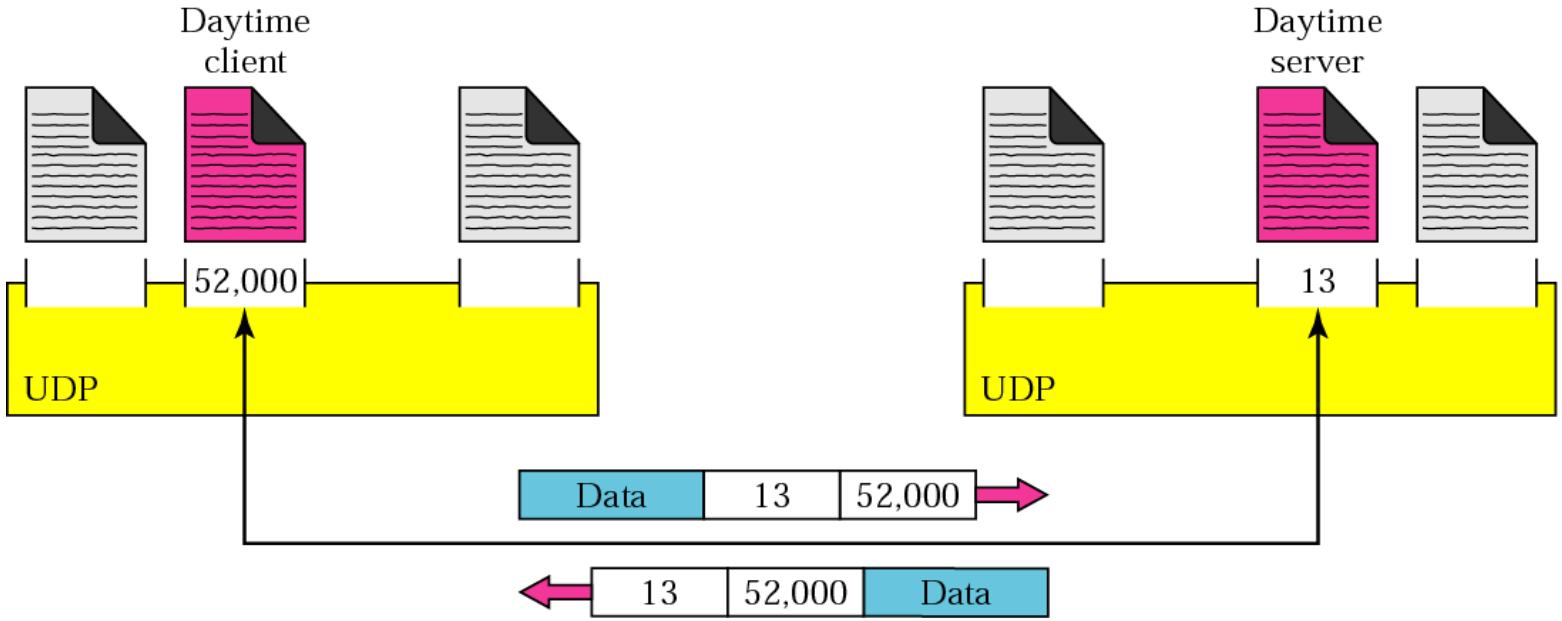
- Serviço TCP:
  - Orientado à ligação: estabelecimento de ligação necessário
  - Transporte fiável das mensagens trocadas
  - Controlo de fluxo: emissor não pode sobrecarregar o receptor
  - Controlo de congestão: emissor não pode sobrecarregar a rede
  - Não fornece: *timestamps*, garantias de largura de banda, segurança

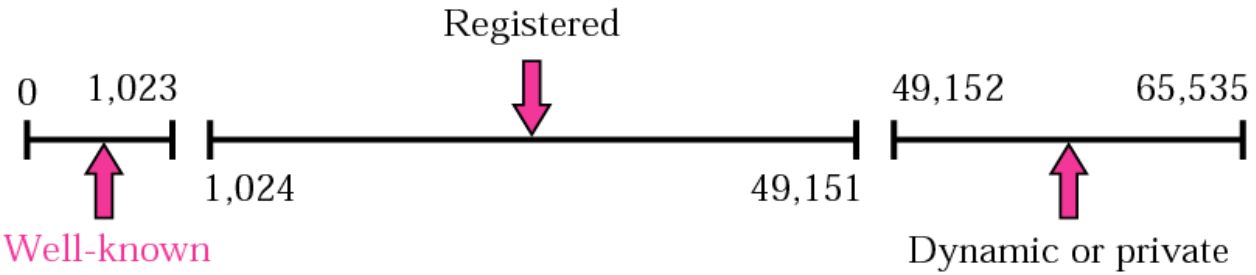
- Serviço UDP:
  - Transferência não fiável de dados entre processos
  - Não fornece: estabelecimento de ligação, fiabilidade, controlo de fluxo ou de congestão, *timestamps*, garantias de largura de banda, segurança

Protocolos de transporte das aplicações

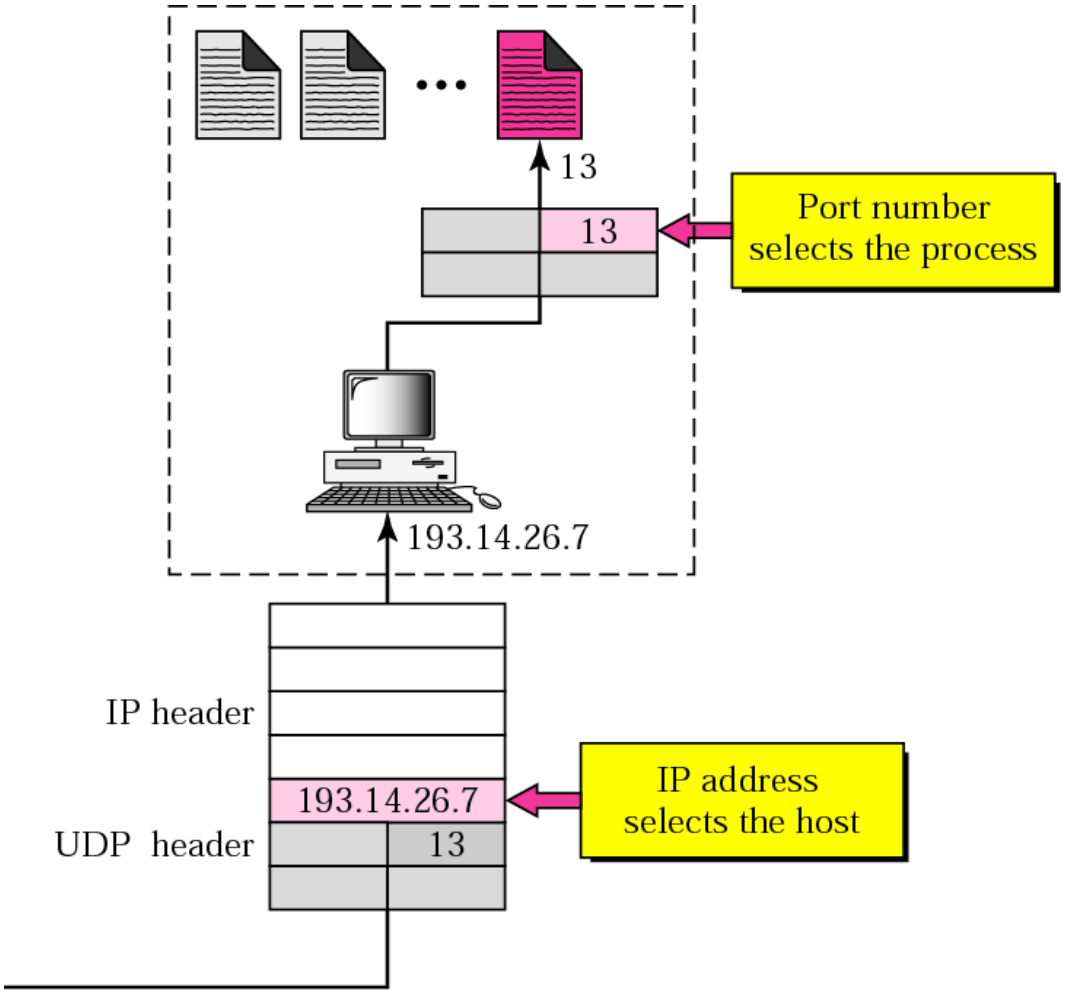
Aplicação	Protocolo da camada de aplicação	Protocolo de transporte
E-mail	SMTP	TCP
Acesso remoto	Telnet, SSH	TCP
WWW	HTTP	TCP
Transferência de ficheiros	HTTP, FTP	TCP
Distribuição vídeo e/ou áudio	HTTP, RTP	TCP ou UDP
Voz sobre IP	SIP, RTP, proprietário	UDP

Conceito de porto (UDP)



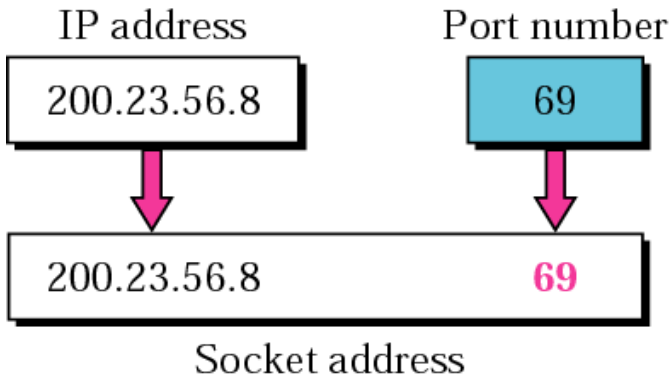


Endereço IP e Número do Porto



Introdução aos sockets

- Socket: interface controlada pelo sistema operativo, criada pela aplicação para que processos possam enviar/receber mensagens de/para outros processos
  - Um socket é constituído por 2 identificadores **[Endereço IP, Porto]**



- Objectivo: aprender a construir aplicações cliente/servidor que comunicam através de sockets
- Socket API:
  - Introduzida no BSD4.1 UNIX, 1981
  - Explicitamente criada para ser utilizada pelas aplicações
  - Paradigma cliente-servidor
  - Dois tipos de transporte estão disponíveis: UDP, TCP

Tarefas cliente/servidor em TCP/IP

- O servidor deve estar pronto para aceitar as ligações dos clientes

- Cliente:

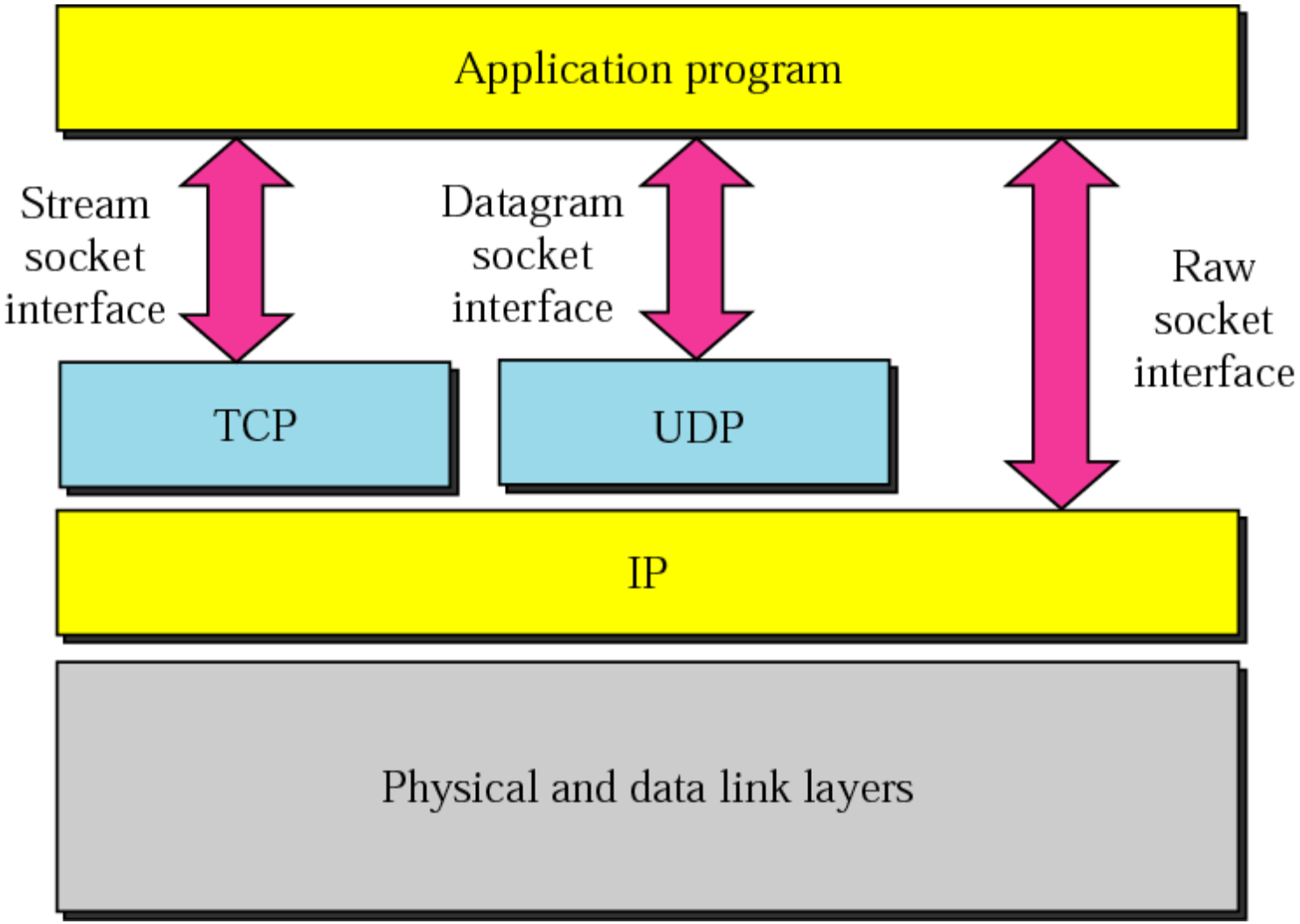
1. Criar um socket TCP
2. Estabelecer ligação
3. Comunicar
4. Fecho da ligação

- Servidor:
  - Criar um socket TCP
  - Atribuir um porto ao socket
  - Colocar o socket à escuta
  - Repetidamente:
    - Aceitar uma nova ligação
    - Comunicar
    - Fechar a ligação

Principais funções da API

Função	Descrição
socket	Cria um novo socket/descritor para comunicação
connect	Iniciar ligação com servidor
send	Escreve dados numa ligação
recv	Lê dados de uma ligação
close	Fecha uma ligação
bind	Atribui um endereço IP e um porto a um socket (associação)
listen	Coloca o socket em modo passivo à escuta na porta
accept	Bloqueia o servidor á espera de uma ligação
recvfrom	Recebe um datagrama e guarda o endereço do emissor
sendto	Envia um datagrama especificando o endereço

Tipos de sockets



Estruturas importantes

Generic

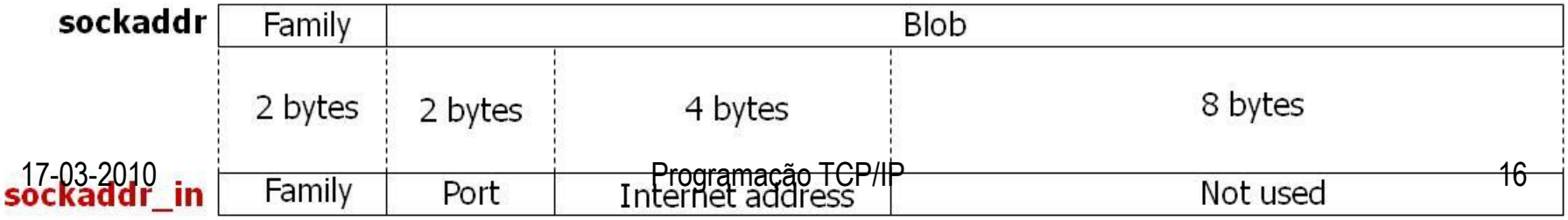
- struct sockaddr

```
{
        unsigned short sa_family;       /* Address family (e.g., AF_INET) */
        char sa_data[14];               /* Protocol-specific address information */
};
```

IP Specific

- struct sockaddr_in

```
{
        unsigned short sin_family;      /* Internet protocol (AF_INET) */
        unsigned short sin_port;        /* Port (16-bits) */
        struct in_addr sin_addr;        /* Internet address (32-bits) */
        char sin_zero[8];               /* Not used */
};
struct in_addr
{
         unsigned long s_addr;          /* Internet address (32-bits) */
};
```



Criar um socket

- mySock = socket(int *family*,int *type*,int *protocol*);
  - Retorna um descritor do socket; -1 em caso de erro
  - Reserva os recursos necessários para a comunicação (*endpoint*)
  - *family*: especifica a família de protocolos, e.g. IPv4 ou IPv6
  - *type*: especifica o tipo de serviço, e.g. stream, datagram ou raw
  - *protocol*: indica o protocolo de transporte
    - O valor 0 funciona como defeito para um dado *type*

	Family	Type	Protocol
TCP	AF_INET (IPv4)	SOCK_STREAM	IPPROTO_TCP
UDP		SOCK_DGRAM	IPPROTO_UDP

Atribuir um endereço a um socket

- int bind(int *socket*, struct sockaddr * *addr*, int *len*);
  - A função bind() é utilizada para atribuir um endereço/porto a um socket existente
  - bind retorna 0 em caso de sucesso ou -1 em caso de erro
  - socket: descritor do socket; addr: estrutura endereço/porto; len: sizeof(addr)
- Exemplo:
  - struct sockaddr_in my_addr;
  - int sockfd = socket(PF_INET, SOCK_STREAM, 0); // retorna descritor do socket
  - my_addr.sin_family = AF_INET; //IPv4
  - my_addr.sin_port = htons(MYPORT); // Porto (network byte order)
  - my_addr.sin_addr.s_addr = inet_addr("10.12.110.57"); // endereço IPv4
  - memset(my_addr.sin_zero, '\0', sizeof my_addr.sin_zero); // os ultimos 8 bytes têm de ser iguais a zero
  - bind(sockfd, (struct sockaddr *)&my_addr, sizeof my_addr); // retorna -1 em caso de erro, 0 caso tenha sucesso

Função listen()

- int *status* = listen(int *socket*, int *queuelen*):
  - Permite aos servidores aceitarem ligações a efectuar
  - Coloca o socket num modo passivo pronto a aceitar ligações
  - O sistema operativo coloca numa fila os múltiplos pedidos simultâneos que sejam recebidos por um socket
  - Apenas válido em sockets com serviço de entrega robusto TCP (stream)

  - status: 0 se estiver a ouvir, -1 em caso de erro
  - socket: descritor do socket
  - queuelen: # de ligações activas que podem ficar pendentes (numa fila) por este socket

Função connect()

- int connect(int *socket*, struct sockaddr **servaddr*, int *addrlen*):
  - Associa um destino permanente a um socket
  - Altera o estado do socket: 'unconnected' -> 'connected'
  - A estrutura sockaddr deve conter um endereço IP/porto da ligação desejada
  - A semântica do connect depende do protocolo de transporte:
    - Ligação TCP : inicia o esatebelecimento de ligação
    - Ligação UDP: cria uma associação entre o socket e o endereço e porto destino. Desta forma dados enviados futuramente vão sempre para este destino
  - status: 0 em caso de sucesso, -1 caso contrário
  - socket: socket a ser utilizado na ligação
  - servaddr: endereço do participante passivo (servidor)
  - addrlen: sizeof(servaddr)

Função accept()

- int *s1* = accept(int *s2*, struct sockaddr * *clientaddr*, int * *addrlen*)
  - Necessita de esperar por uma ligação
  - Bloqueia até que o pedido de ligação chegue
  - Quando um pedido chega, é preenchido o argumento *addr* com o endereço do cliente que efectuou o pedido e *addrlen* contém o comprimento do endereço
  - Também é criado um novo socket, retorna-se o seu descritor

  - s1: o novo socket (a usar para transferência de dados)
  - s2: o socket original (utilizado para ouvir o porto)
  - clientaddr: struct sockaddr, contém o endereço do cliente (efectua filtragem)
  - namelen: sizeof(name): value/result parameter

TCP: enviando/recebendo dados

- int count = send(int *socket*, const char **msg*, int len, int *flags*)
  - Socket com ligação realizada, flags controlam a transmissão, e.g. dados URG
- int count = recv(int *socket*, char **msg*, int *len*, int *flags*)
  - Socket com ligação realizada, flags controlam a recepção, e.g. look-ahead
- Parâmetros:
  - count: # bytes transmitidos (-1 no caso de erro)
  - socket: descritor do socket
  - msg: buffer a transmitir/receber, len: comprimento do buffer msg em bytes
  - flags: inteiro, opções especiais, habitualmente assume o valor 0
- Para enviar/receber dados, as seguintes funções (de escrita e leitura ficheiros) também podem ser usadas:
  - ssize_t read(int *filedes*, void* *buf*, size_t *nbytes*)
  - ssize_t write(int *filedes*, const void* *buf*, size_t *nbyte*)

UDP: enviando/recebendo dados

- Sem estabelecimento de ligação (SOCK_DGRAM):
  - int *count* = sendto(int *socket*, const void* *buf*, int *len*, int *flags*, const struct sockaddr* *destaddr*, int *addrlen*);
    - count, sock, buf, len, flags: o mesmo que o send()
    - destaddr: endereço do destino
    - addrlen: sizeof(destaddr)
  - int *count* = recvfrom(int *socket*, void **buf*, int *length*, int *flags*, struct sockaddr *src*addr*, int * *addrlen*);
    - count, sock, buf, len, flags: as mesmas que o recv()
    - srcaddr: endereço da origem
    - addrlen: sizeof(srcaddr)

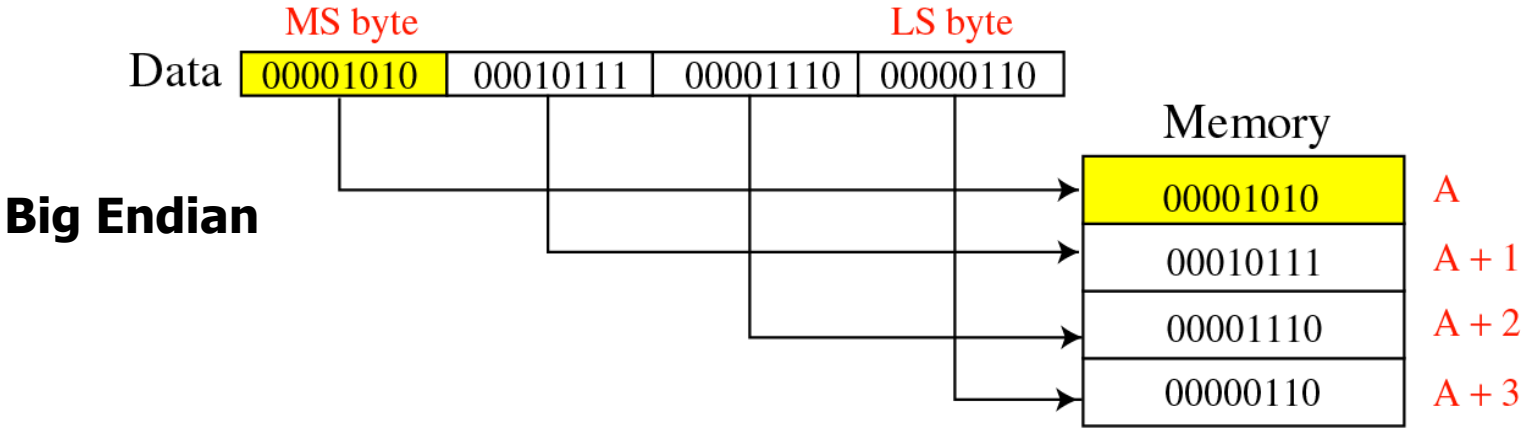
close(), shutdown()

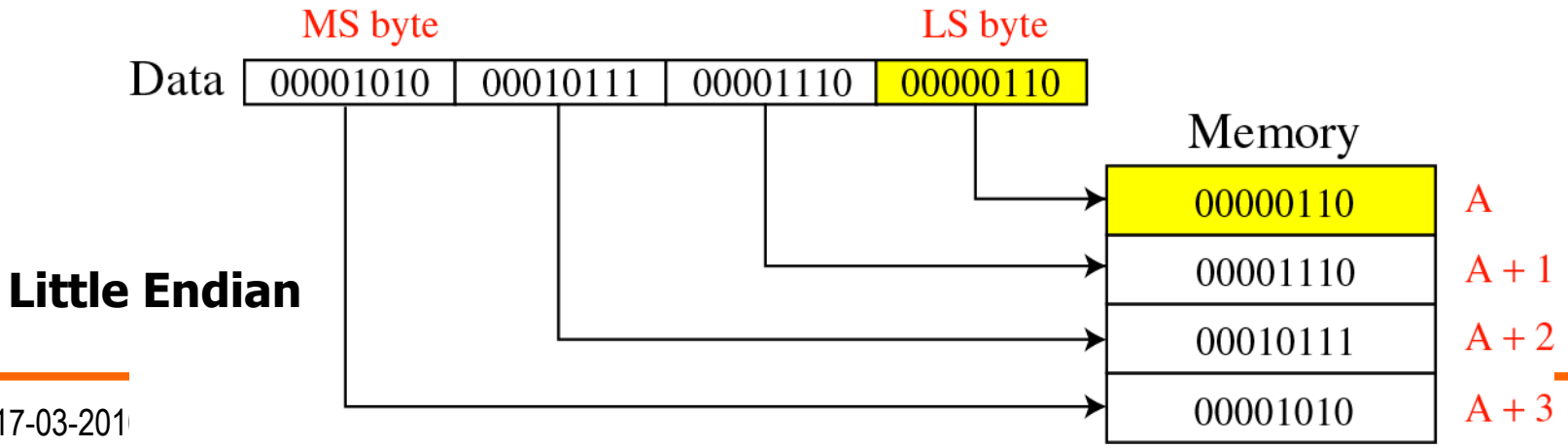
- close(int *socket*)
  - Para sockets UDP, o fecho irá libertar os recursos reservados no porto local associado a este socket
  - Para sockets TCP, isto irá iniciar uma troca de mensagens entre máquinas para fechar a ligação antes de libertar os recurso do *socket*
  - O TCP irá tentar enviar os dados na fila de espera antes da sequência de fecho

- shutdown(int *socket*, int *how*)
  - Se *how=0*, não será permitido mais nenhuma leitura (recv) do *socket*
  - Se *how=1*, escritas posteriores (send) não serão permitidas. O *socket* terá ainda de ser fechado.

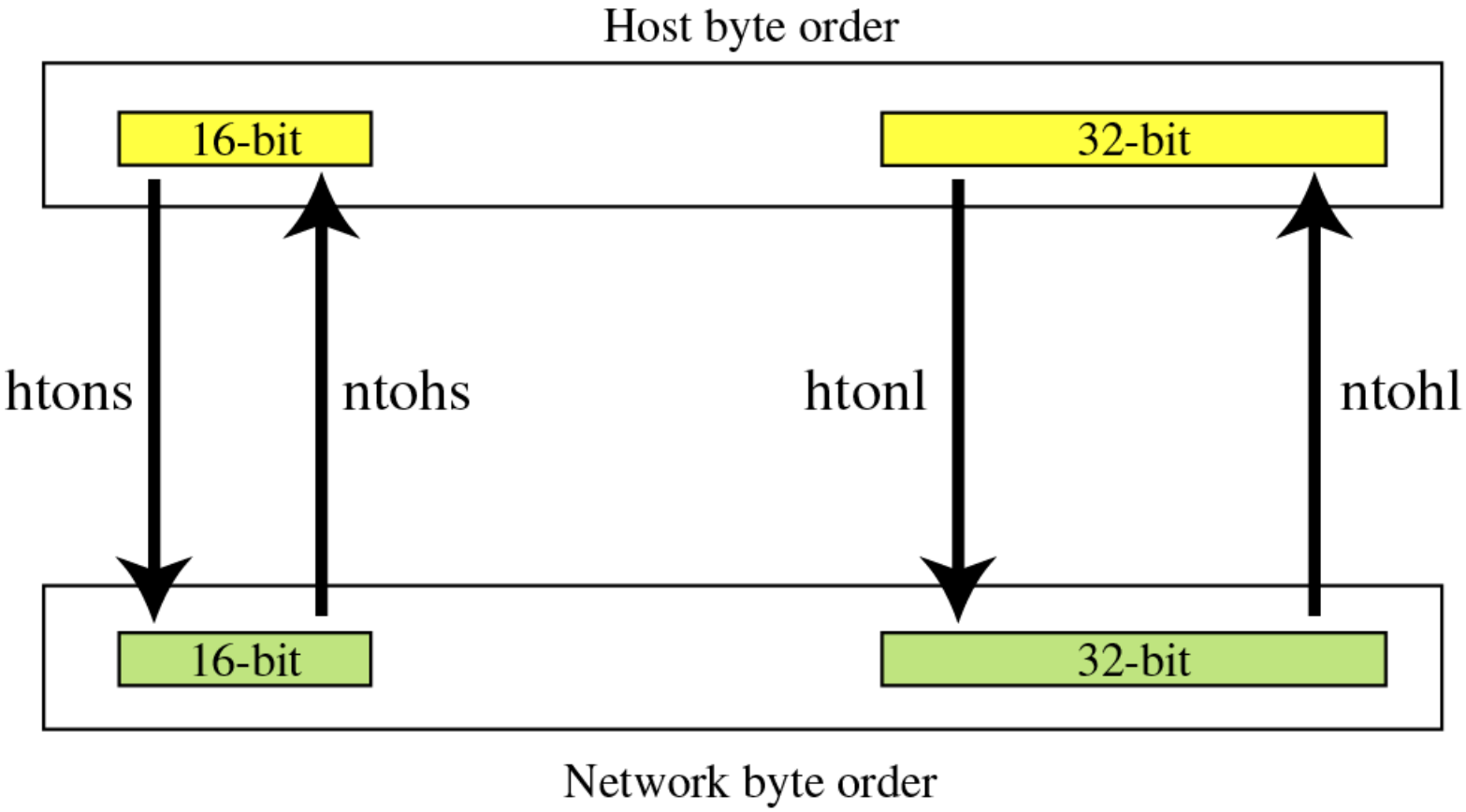
Byte order

- A ordenação dos bytes depende da arquitectura da máquina:
  - Intel: little-endian, Sparc, PowerPC: big-endian, **TCP/IP order: big-endian**





Funções de ordenação de bytes



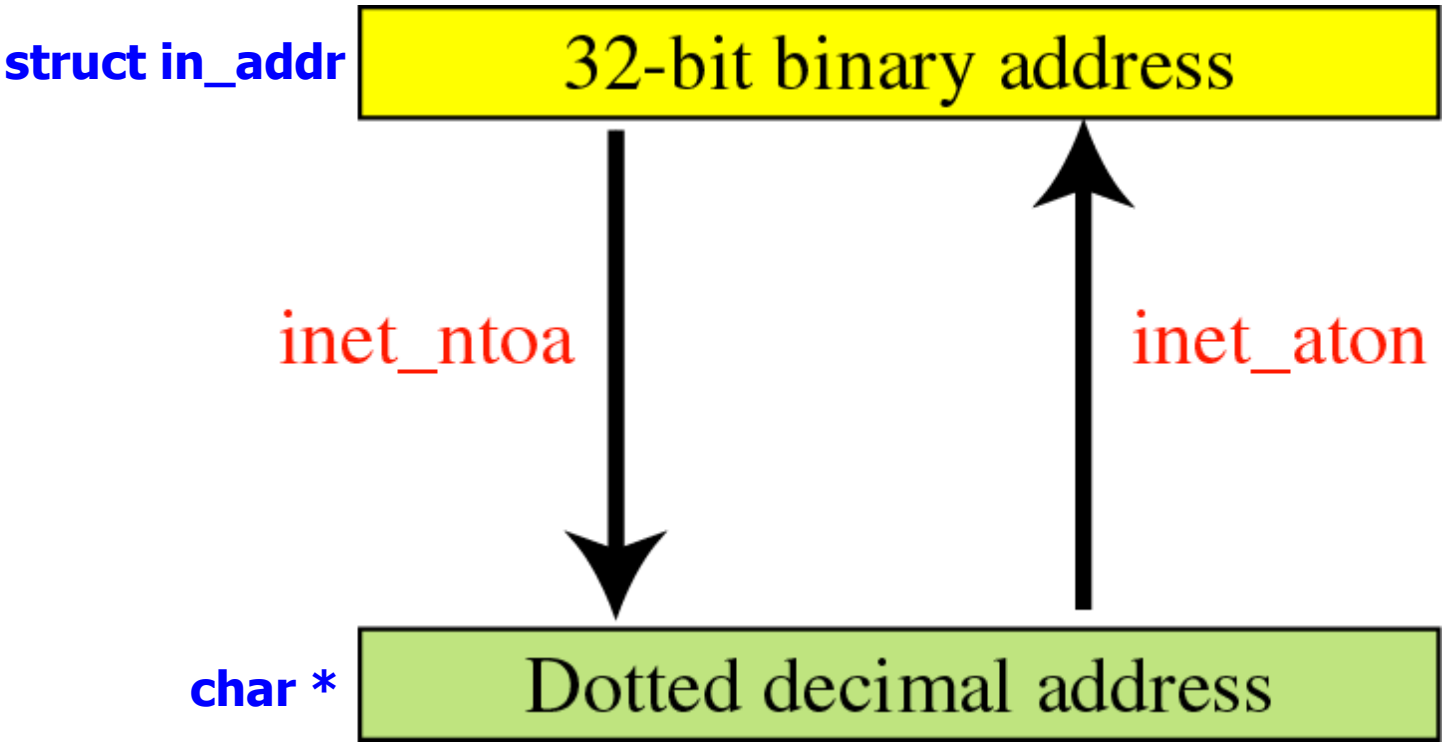
Funções de ordenação de bytes

- Funções:
  - **u_long m =ntohl(u_long m)**
    - network-to-host byte order, 32 bit—
  - **u_long m =htonl(u_long m)**
    - host-to-network byte order, 32 bit—
  - **ntohs(),htons()**
    - short (16 bit)

Exemplo:

```
struct sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_port = htons(9999);
sin.sin_addr.s_addr = inet_addr;
```

Transformação de endereços



Transformação de endereços

- **char *inet_ntoa(struct in_addr *ip*)**
  - Converte o valor ***ip*** de 32-bits (network byte order) para uma string (char*)
  - O ponteiro obtido por *inet_ntoa* contém a string. Deve-se copiar a *string* para outro buffer antes de chamar a função de novo.
- **unsigned int inet_addr(char *str)**
  - O valor str representa um endereço IP (na notação X.X.X.X); inet_addr deve retornar um inteiro de 32-bit (network byte order).
  - Este valor pode ser usado para o campo sin_addr.s_addr de uma estrutura socketaddr_in:
    - Ex.: sin_addr = inet_addr(“152.14.51.129”);
  - O valor -1 é obtido em caso de erro

Obter informação acerca das máquinas

- struct hostent *hptr; /*includes host address in binary*/
- hptr=gethostbyname(char *name);
  - Exemplo:gethostbyname(“www.csc.ncsu.edu”);

- struct hostent *hptr;
- hptr=gethostbyaddr(char *addr,int addrlen, int addrtype);
  - Exemplo: gethostbyaddr(&addr, 4, AF_INET);

- struct servent *sptr; /* includes port and protocol */
- sptr=getservbyname(char *name, char *proto);
  - Exemplo: getservbyname(“smtp”, “tcp”);

- struct protoent *pptr; /* includes protocol number */
- pptr=getprotobyname(char *name);
  - Exemplo: getprotobyname(“tcp”);

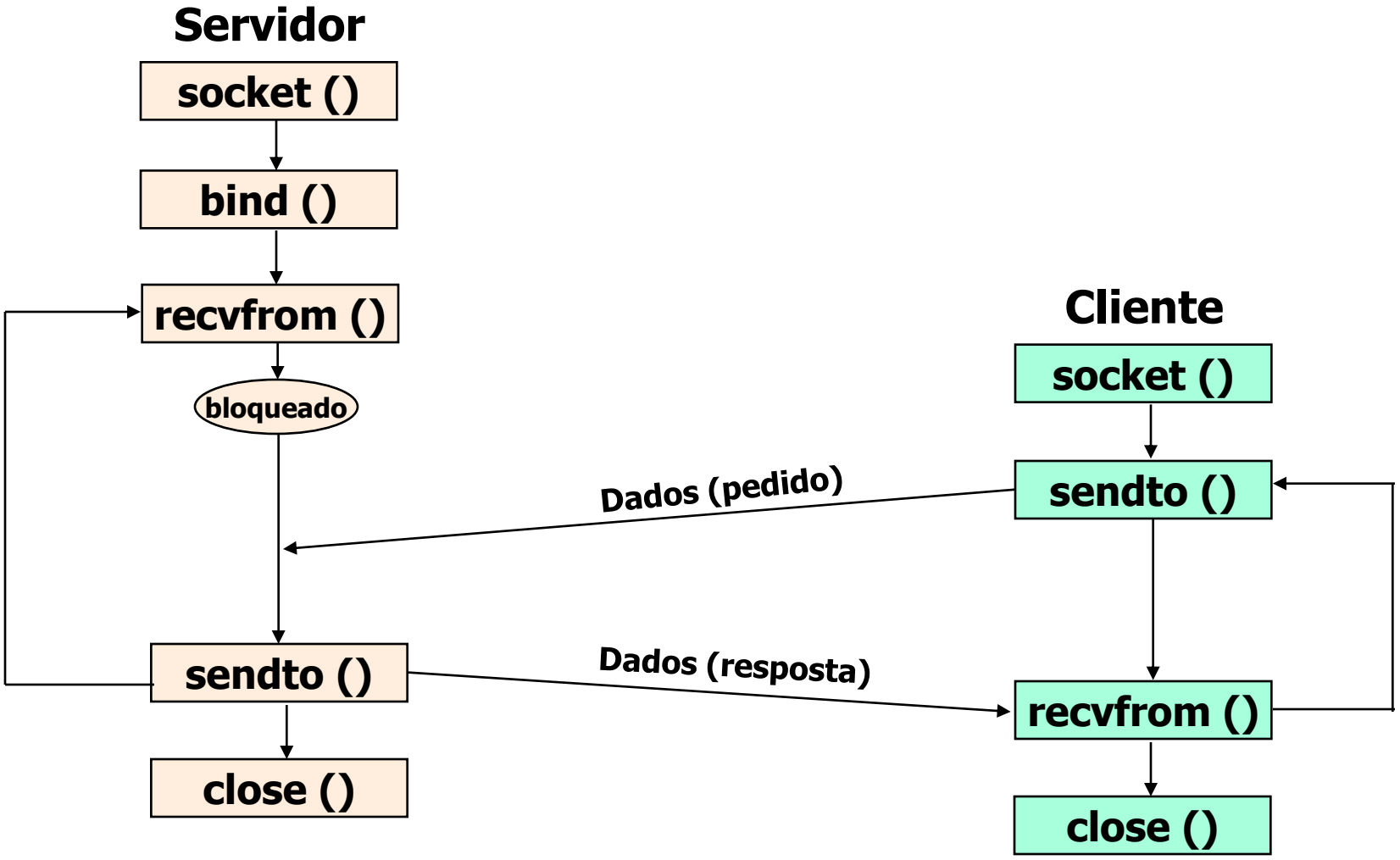
Sockets: fundamentos

- O servidor deve estar a activo antes do cliente poder enviar quaisquer dados
- O servidor deve ter um socket através do qual recebe e envia as mensagens
- O socket é localmente identificado com um número de porto
- Cliente precisa de conhecer o endereço IP e o número do porto associado ao socket

Programação de sockets com UDP

- UDP: nenhuma “ligação” entre cliente e servidor
  - O emissor anexa explicitamente o endereço IP e o porto de destino a cada datagrama
  - O sistema operativo anexa o endereço IP e o porto do socket de envio a cada datagrama
    - Não é necesário fazer bind() quando se envia informação
  - A partir do segmento recebido, o servidor pode extrair o endereço IP e porto do emissor
    - Se receber informação é preciso fazer bind()
    - Nota: as mensagens enviadas através do UDP são oficialmente chamadas “datagramas”

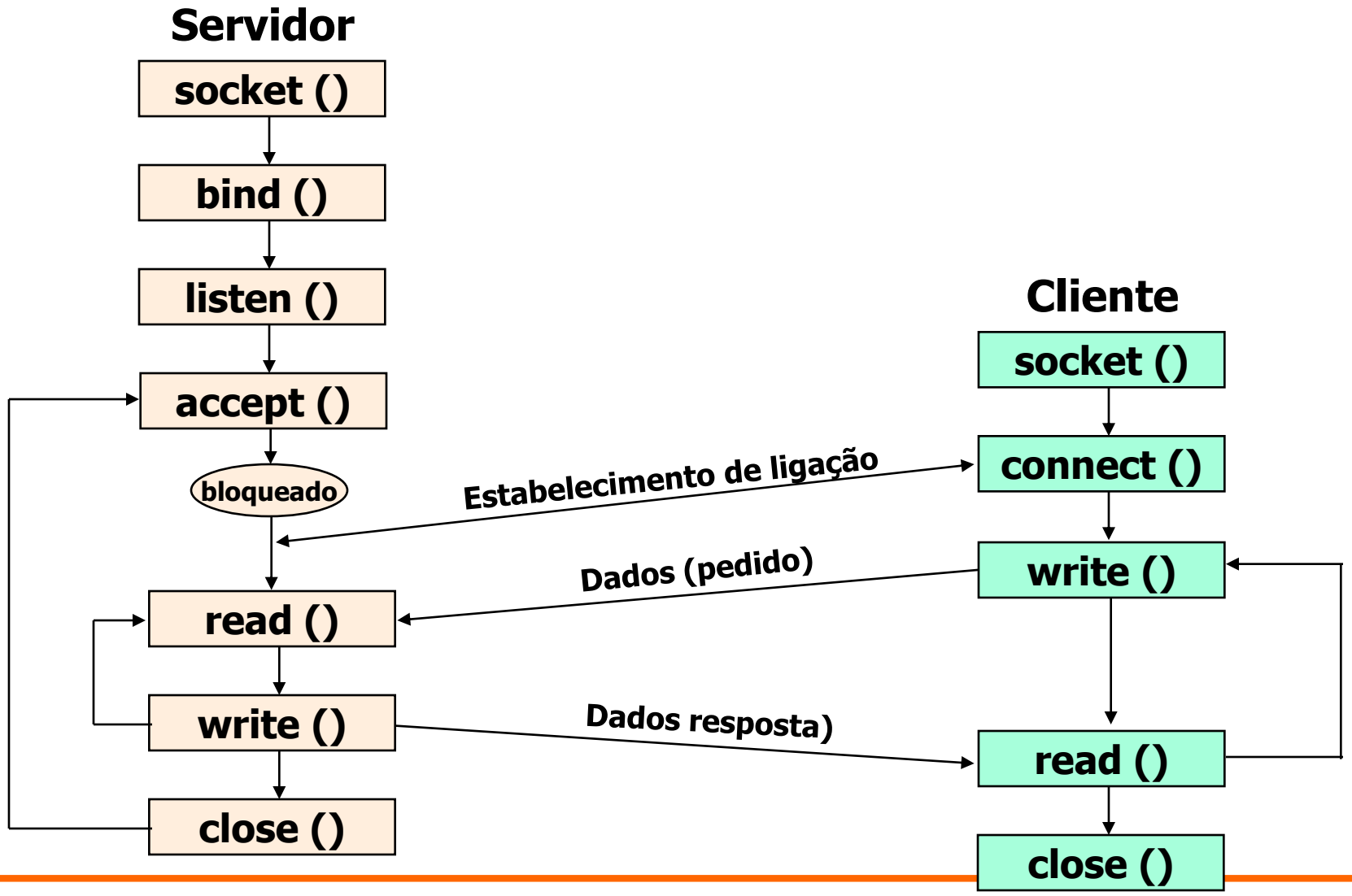
Comunicação sem ligação (UDP)



Programação de sockets com TCP

- O cliente deve contactar o servidor:
  - O processo do servidor tem de estar a ser executado
  - O servidor criou préviamente um socket para receber a ligação do cliente
- O cliente contacta o servidor através:
  - Da criação de um socket TCP local ao cliente
  - Especificando o endereço IP e o número do porto do processo do servidor
  - Quando um cliente cria um socket: o cliente TCP estabeleçe uma ligação ao servidor TCP
- Quando contactado pelo cliente, o servidor TCP cria um novo socket para o processo do servidor comunicar com cada cliente
  - Permite ao servidor 'falar' com múltiplos clientes
  - Números de portos de origem utilizados para distinguir os clientes

Comunicação com ligação (TCP)



```
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
...

int fd, newfd;
struct hostent *hostptr;
struct sockaddr_in serveraddr, clientaddr;
int clientlen;
...
```

TCP Server

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

memset((void*)&serveraddr,(int)'\0',  sizeof(serveraddr));
serveraddr.sin_family=AF_INET;
serveraddr.sin_addr.s_addr=htonl(INADDR_ANY);
serveraddr.sin_port=htons((u_short)58000);

bind(fd,(struct sockaddr*)&serveraddr, sizeof(serveraddr));

listen(fd,5);

clientlen=sizeof(clientaddr);
newfd=accept(fd,(struct sockaddr*)&clientaddr,
                &clientlen);
```

blocks until connection from client

TCP Client

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

hostptr=gethostbyname("tejo.isel.ipl.pt");

memset((void*)&serveraddr,(int)'\0',
      sizeof(serveraddr));
serveraddr.sin_family=AF_INET;
serveraddr.sin_addr.s_addr=((struct in_addr *)
   (hostptr->h_addr_list[0]))->s_addr;
serveraddr.sin_port=htons((u_short)58000);

connect(fd,(struct sockaddr*)&serveraddr,
       sizeof(serveraddr));
```

connection establishment

TCP three-way handshake

```
write(fd,...);
...
read(fd,...);
...
close(fd);
```

```
read(newfd,...);
...
write(newfd,...);
...


close(fd); close(newfd);
```

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
...

int fd;
struct hostent *hostptr;
struct sockaddr_in serveraddr, clientaddr;
int addrlen;
...
```

UDP Server

```
fd=socket(AF_INET,SOCK_DGRAM,0);

memset((void*)&serveraddr,(int)'\0',
                sizeof(serveraddr));
serveraddr.sin_family=AF_INET;
serveraddr.sin_addr.s_addr=htonl(INADDR_ANY);
serveraddr.sin_port=htons((u_short)58000);

bind(fd,(struct sockaddr*)&serveraddr,
      sizeof(serveraddr));

addrlen=sizeof(clientaddr);
recvfrom(fd,buffer,sizeof(buffer),0,
       (struct sockaddr*)&clientaddr,
       &addrlen);
```

blocks until datagram received from a client

```
...
sendto(fd,msg,strlen(msg)+1,0,
      (struct sockaddr*)&clientaddr,addr_len);
...

close(fd);
```

UDP Client

```
fd=socket(AF_INET,SOCK_DGRAM,0);

hostptr=gethostbyname("tejo.isel.ipl.pt");

memset((void*)&serveraddr,(int)'\0',
      sizeof(serveraddr));
serveraddr.sin_family=AF_INET;
serveraddr.sin_addr.s_addr=((struct in_addr *)
   (hostptr->h_addr_list[0]))->s_addr;
serveraddr.sin_port=htons((u_short)58000);

addrlen=sizeof(serveraddr);

sendto(fd,msg,strlen(msg)+1,0,
      (struct sockaddr*)&serveraddr,addrlen);
...
addrlen=sizeof(serveraddr);
recvfrom(fd,buffer,sizeof(buffer),0,
       (struct sockaddr*)&serveraddr,&addrlen);
...

close(fd);
```

Referências

- “Computer Networking, a top down approach featuring the Internet (4th edition)”, James F. Kurose (Author), Keith W. Ross, Addison-Wesley Longman.

- “TCP/IP Protocol Suite”, Behrouz A. Forouzan, Sophia C. Fegan, McGraw-Hill Professional.