

Sockets (BSD)

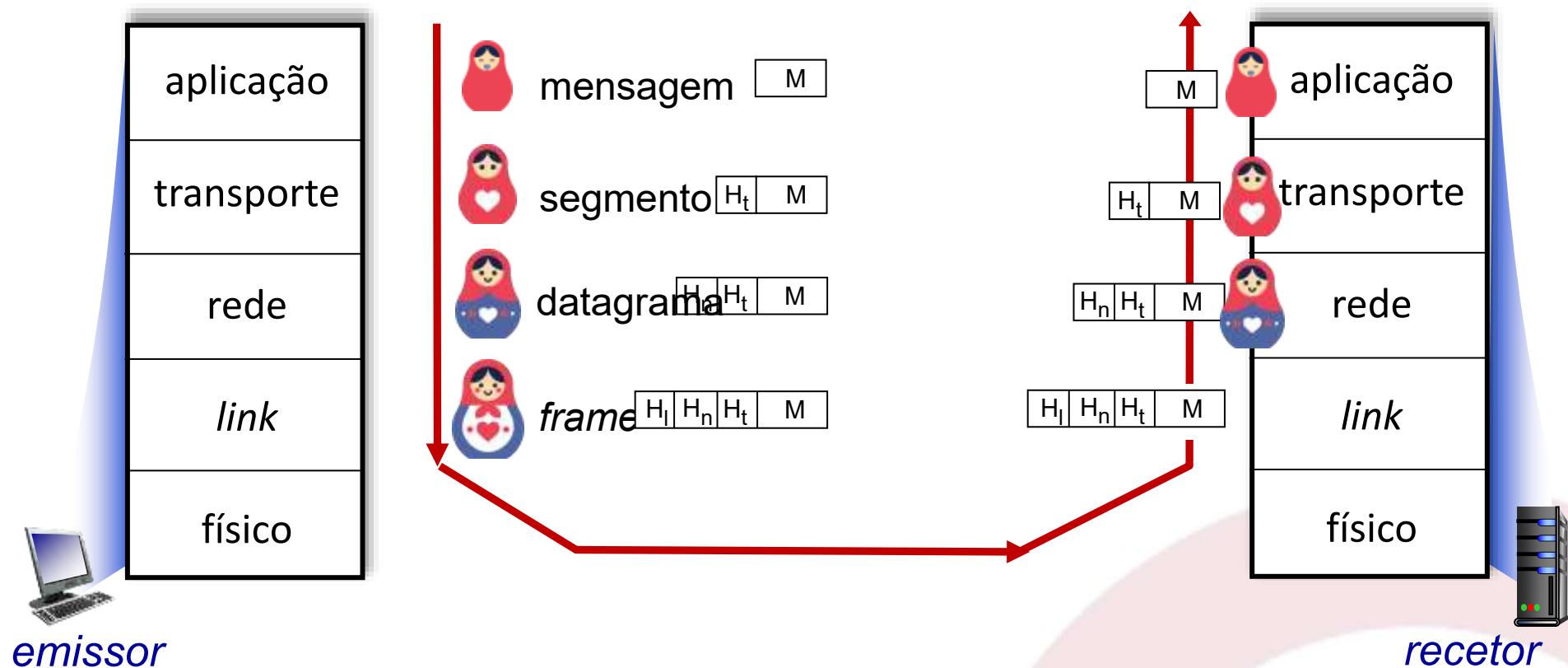
Patrício Domingues

(c) Patrício Domingues, Vitor Carreira



SOCKETS

Serviços, camadas e encapsulamento

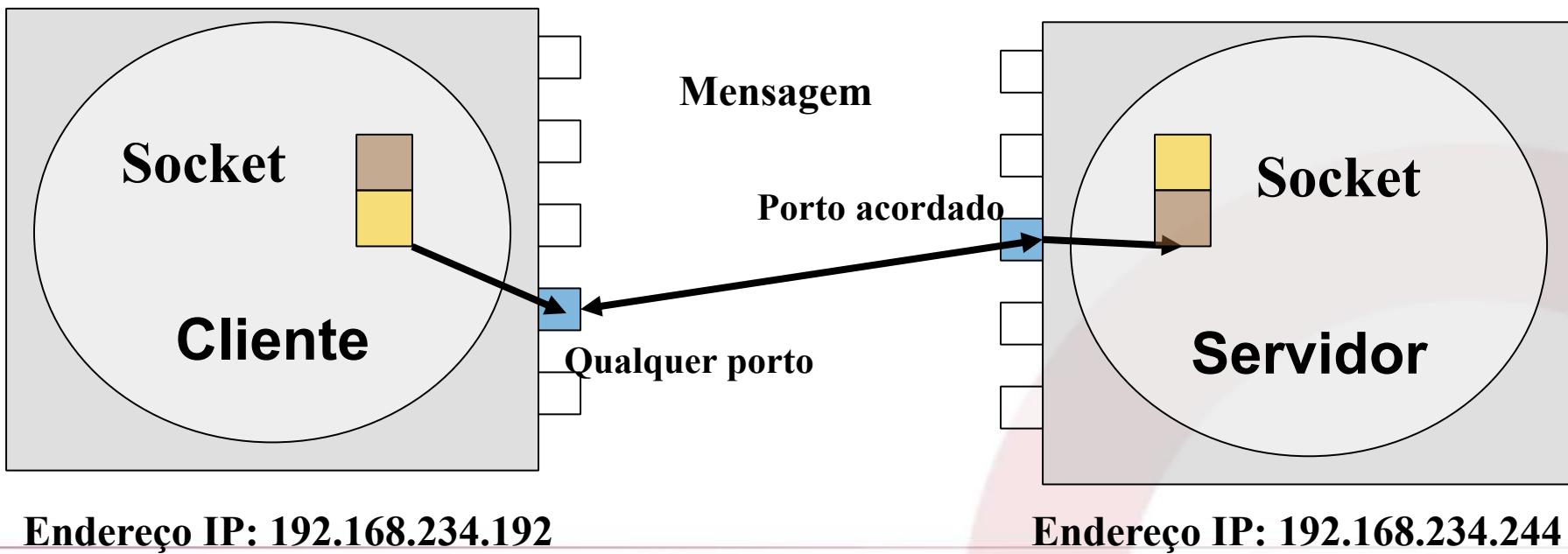


Adaptado de:

Computer Networking: A Top-Down Approach
8th edition, Jim Kurose, Keith Ross, Pearson, 2020

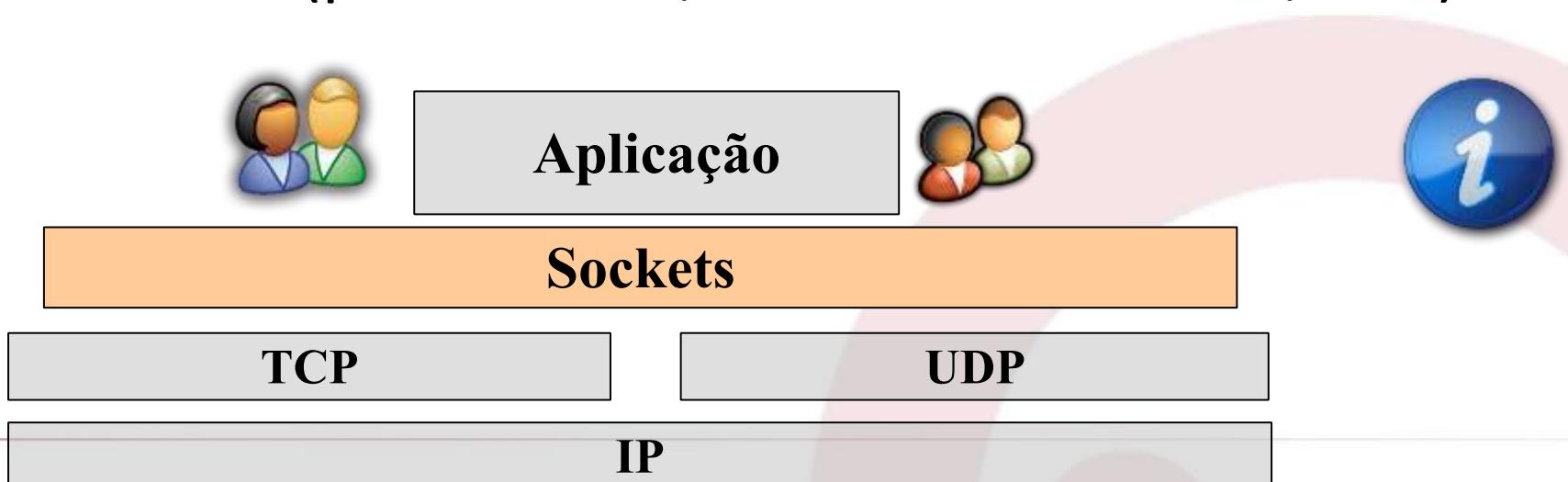
Camada de transporte

- Nível de processos
 - Dois processos em máquinas distintas trocam pacotes (segmentos TCP ou datagramas UDP)
 - IP + porto origem ← – protocolo – → IP + porto destino



Sockets

- APIs de programação da camada de transporte
 - Interface entre a camada da aplicação e a camada de transporte
 - Possibilita o acesso e uso dos serviços da camada de transporte (TCP e UDP) às aplicações
 - Consiste num conjunto de funções com comportamento bem definido (parâmetros, valores de retorno, etc.)



- O TCP/IP não inclui nenhuma definição de *Application Programming Interface* (API)
- Existem várias API para a programação de aplicações sobre a pilha protocolar TCP/IP
 - **Sockets (UNIX Sockets, Java Sockets, Winsock)**
 - XTI (Evolução TLI)
 - XTI – X/Open Transport Interface
 - TLI – Transport Layer Interface
 - MacTCP (Apple)
 - Obsoleto, substituída pela API OpenTransport (1995)
 - OpenTransport (Apple)
 - Obsoleto, substituído pela API BSD



Interface Socket BSD (1)

- **Socket**
 - Encaixe (tradução)

Wall socket
- **No que respeita à comunicação distribuída**
 - Socket
 - Mecanismo de comunicação entre processos sejam eles do mesmo sistema (comunicação local) ou de diferentes sistemas (comunicação remota)
 - Canal de comunicação bidireccional entre dois ou mais processos
 - Encaixa no nível 4 do modelo OSI da ISO (camada de transporte)
- **Interface Socket BSD (Berkeley Software Distribution)**
 - API *standard* para programar a pilha protocolar TCP/IP
 - Vantagens da interface Socket BSD
 - Portabilidade das aplicações
 - Simplificação do desenvolvimento de aplicações



- Genérica

- Suporte para múltiplas famílias de protocolos (TCP, UDP, etc.; IPv4, IPV6, etc.)
- Representação genérica de endereços (*struct sockaddr*)

- Promove a utilização da interface de programação de Entrada/Saída (I/O)

- Algumas funções partilhadas com API de ficheiros baixo nível
 - write, read, close
 - ...

Endereçamento através de sockets

- Uma ligação socket é plenamente descrita através de cinco parâmetros
 - Protocolo de transporte (UDP/TCP/...)
 - Endereço local
 - Porto local
 - Endereço remoto
 - Porto remoto
- Exemplo
 - (TCP, 192.168.234.21, 4500, 192.168.234.7, 22)

Conceito de porto

Do not
forget!

■ Porto

- Identificador numéricico (inteiro) com 16 bits
 - 16 bits → existem 2^{16} (65536) portos diferentes
 - Deste modo, uma aplicação é identificada pelo:
 - endereço IP
 - porto (1 a 65535)
 - Um sistema pode manter várias ligações em simultâneo com outro sistema
 - IP + porto

■ Analogia

- “Porto” é como uma extensão de uma central telefónica
 - IP é como o número de telefone central

COMMON PORTS		TCP/UDP Port Numbers		packetlife.net	
7 Echo	514 STUP	2740 Telnet M	6963-6965	Instant Chat	
18 Chattrm	548-547 SICN/CHAT	2867 Tomcat/SSH	6978	Qmailbox	
20-21 TFTP	566 vncserver	2898 telnetd/CR	7212	CloudOffice	
22 SSH	512 SSH	2914 WWW WWW	7449-7549	CloudOffice	
23 Telnet	507 TFTP	3014 WWW WWW	8000-8010	CloudOffice	
25 SMTP	503 TELNET	3027 WWW WWW	8010-8020	CloudOffice	
42 DNS/Registration	509 RSH/RDYN/SCOM	3029 TFTP proxy	8020-8030	CloudOffice	
43 WHOIS	533 reverse ARPing	3030 QLLP	8030-8040	CloudOffice	
49 TACACS	516 TACACS TACACS	3046-3054 TFTP	8040-8050	CloudOffice	
53 DNS	519 DNS DNS	3055 QLLP	8050-8060	CloudOffice	
64-65 POP3/POP3S	520 POP3 POP3S	3056 WWW WWW	8060-8070	CloudOffice	
69 TFTP	561 TFTP TFTP	3057 WWW WWW	8070-8080	CloudOffice	
70 Systat	565 Systat Systat	3058 WWW WWW	8080-8090	CloudOffice	
79 Finger	573 Nntp	3164-3165 WWW WWW	8090-8100	CloudOffice	
80 HTTP	582 VNC/VNC Server	3166-3167 WWW WWW	8110-8120	CloudOffice	
88 Kerberos	588 H2C	3168-3169 WWW WWW	8120-8130	CloudOffice	
90 IM/Exchange	593 IM/Exchange IM/Exchange	3170-3171 WWW WWW	8130-8140	CloudOffice	
110 IMAP	595 IMAP IMAP	3172-3173 WWW WWW	8140-8150	CloudOffice	
113 IMAP	597 IMAP IMAP	3174-3175 WWW WWW	8150-8160	CloudOffice	
119 NNTP-Standard	1220-1225 NNTP NNTP	3176-3177 WWW WWW	8160-8170	CloudOffice	
123 NNTP-SSL	1226-1229 NNTP NNTP	3178-3179 WWW WWW	8170-8180	CloudOffice	
139 Microsoft RPC	1389 RPC RPC	3180-3181 WWW WWW	8180-8190	CloudOffice	
143-145 IMAP	1400 IMAP IMAP	3182-3183 WWW WWW	8190-8200	CloudOffice	
147-163 IMAP	1413 IMAP IMAP	3184-3185 WWW WWW	8200-8210	CloudOffice	
177 JDCE/J	1543 JDBC JDBC	3186-3187 WWW WWW	8210-8220	CloudOffice	
179 IMAP	1557 IMAP IMAP	3188-3189 WWW WWW	8220-8230	CloudOffice	
205 AppleTalk	1429-1434 Microsoft SQL	3190-3191 WWW WWW	8230-8240	CloudOffice	
204 BEEP	1813 WINS WINS	3192-3193 WWW WWW	8240-8250	CloudOffice	
218 TSP	1828 Cisco VTP	3194-3195 WWW WWW	8250-8260	CloudOffice	
302-303 RIP (distance)	1951 RIP RIP	3196-3197 WWW WWW	8260-8270	CloudOffice	
305 UDP	1973 IGMP IGMP	3198-3199 WWW WWW	8270-8280	CloudOffice	
511-612 Direct Connect		3200-3201 WWW WWW	8280-8290	CloudOffice	
443 HTTPS/TLS		3202-3203 WWW WWW	8290-8300	CloudOffice	
445 Microsoft DFS		3204-3205 WWW WWW	8300-8310	CloudOffice	
465 Outlook		3206-3207 WWW WWW	8310-8320	CloudOffice	
505 Outlook Express		3208-3209 WWW WWW	8320-8330	CloudOffice	
506 Outlook Express		3210-3211 WWW WWW	8330-8340	CloudOffice	
512 Outlook		3212-3213 WWW WWW	8340-8350	CloudOffice	
513 Outlook		3214-3215 WWW WWW	8350-8360	CloudOffice	
514 Outlook		3216-3217 WWW WWW	8360-8370	CloudOffice	
515 Outlook Express		3218-3219 WWW WWW	8370-8380	CloudOffice	
520 IMAP		3220-3221 WWW WWW	8380-8390	CloudOffice	
522 POP3		3222-3223 WWW WWW	8390-8400	CloudOffice	
523 Outlook Express		3224-3225 WWW WWW	8400-8410	CloudOffice	
524 Outlook		3226-3227 WWW WWW	8410-8420	CloudOffice	
525 Outlook Express		3228-3229 WWW WWW	8420-8430	CloudOffice	
526 Outlook		3230-3231 WWW WWW	8430-8440	CloudOffice	
527 Outlook Express		3232-3233 WWW WWW	8440-8450	CloudOffice	
540 IMCF		3234-3235 WWW WWW	8450-8460	CloudOffice	
545-546 Outlook		3236-3237 WWW WWW	8460-8470	CloudOffice	
547-548 Outlook		3238-3239 WWW WWW	8470-8480	CloudOffice	
549-550 Outlook		3240-3241 WWW WWW	8480-8490	CloudOffice	
551-552 Outlook		3242-3243 WWW WWW	8490-8500	CloudOffice	
553-554 Outlook		3244-3245 WWW WWW	8500-8510	CloudOffice	
555-556 Outlook		3246-3247 WWW WWW	8510-8520	CloudOffice	
557-558 Outlook		3248-3249 WWW WWW	8520-8530	CloudOffice	
559-560 Outlook		3250-3251 WWW WWW	8530-8540	CloudOffice	
561-562 Outlook		3252-3253 WWW WWW	8540-8550	CloudOffice	
563-564 Outlook		3254-3255 WWW WWW	8550-8560	CloudOffice	
565-566 Outlook		3256-3257 WWW WWW	8560-8570	CloudOffice	
567-568 Outlook		3258-3259 WWW WWW	8570-8580	CloudOffice	
569-570 Outlook		3260-3261 WWW WWW	8580-8590	CloudOffice	
571-572 Outlook		3262-3263 WWW WWW	8590-8600	CloudOffice	
573-574 Outlook		3264-3265 WWW WWW	8600-8610	CloudOffice	
575-576 Outlook		3266-3267 WWW WWW	8610-8620	CloudOffice	
577-578 Outlook		3268-3269 WWW WWW	8620-8630	CloudOffice	
579-580 Outlook		3270-3271 WWW WWW	8630-8640	CloudOffice	
581-582 Outlook		3272-3273 WWW WWW	8640-8650	CloudOffice	
583-584 Outlook		3274-3275 WWW WWW	8650-8660	CloudOffice	
585-586 Outlook		3276-3277 WWW WWW	8660-8670	CloudOffice	
587-588 Outlook		3278-3279 WWW WWW	8670-8680	CloudOffice	
589-590 Outlook		3280-3281 WWW WWW	8680-8690	CloudOffice	
591-592 Outlook		3282-3283 WWW WWW	8690-8700	CloudOffice	
593-594 Outlook		3284-3285 WWW WWW	8700-8710	CloudOffice	
595-596 Outlook		3286-3287 WWW WWW	8710-8720	CloudOffice	
597-598 Outlook		3288-3289 WWW WWW	8720-8730	CloudOffice	
599-600 Outlook		3290-3291 WWW WWW	8730-8740	CloudOffice	
601-602 Outlook		3292-3293 WWW WWW	8740-8750	CloudOffice	
603-604 Outlook		3294-3295 WWW WWW	8750-8760	CloudOffice	
605-606 Outlook		3296-3297 WWW WWW	8760-8770	CloudOffice	
607-608 Outlook		3298-3299 WWW WWW	8770-8780	CloudOffice	
609-610 Outlook		3300-3301 WWW WWW	8780-8790	CloudOffice	
611-612 Outlook		3302-3303 WWW WWW	8790-8800	CloudOffice	
613-614 Outlook		3304-3305 WWW WWW	8800-8810	CloudOffice	
615-616 Outlook		3306-3307 WWW WWW	8810-8820	CloudOffice	
617-618 Outlook		3308-3309 WWW WWW	8820-8830	CloudOffice	
619-620 Outlook		3310-3311 WWW WWW	8830-8840	CloudOffice	
621-622 Outlook		3312-3313 WWW WWW	8840-8850	CloudOffice	
623-624 Outlook		3314-3315 WWW WWW	8850-8860	CloudOffice	
625-626 Outlook		3316-3317 WWW WWW	8860-8870	CloudOffice	
627-628 Outlook		3318-3319 WWW WWW	8870-8880	CloudOffice	
629-630 Outlook		3320-3321 WWW WWW	8880-8890	CloudOffice	
631-632 Outlook		3322-3323 WWW WWW	8890-8900	CloudOffice	
633-634 Outlook		3324-3325 WWW WWW	8900-8910	CloudOffice	
635-636 Outlook		3326-3327 WWW WWW	8910-8920	CloudOffice	
637-638 Outlook		3328-3329 WWW WWW	8920-8930	CloudOffice	
639-640 Outlook		3330-3331 WWW WWW	8930-8940	CloudOffice	
641-642 Outlook		3332-3333 WWW WWW	8940-8950	CloudOffice	
643-644 Outlook		3334-3335 WWW WWW	8950-8960	CloudOffice	
645-646 Outlook		3336-3337 WWW WWW	8960-8970	CloudOffice	
647-648 Outlook		3338-3339 WWW WWW	8970-8980	CloudOffice	
649-650 Outlook		3340-3341 WWW WWW	8980-8990	CloudOffice	
651-652 Outlook		3342-3343 WWW WWW	8990-9000	CloudOffice	
653-654 Outlook		3344-3345 WWW WWW	9000-9010	CloudOffice	
655-656 Outlook		3346-3347 WWW WWW	9010-9020	CloudOffice	
657-658 Outlook		3348-3349 WWW WWW	9020-9030	CloudOffice	
659-660 Outlook		3350-3351 WWW WWW	9030-9040	CloudOffice	
661-662 Outlook		3352-3353 WWW WWW	9040-9050	CloudOffice	
663-664 Outlook		3354-3355 WWW WWW	9050-9060	CloudOffice	
665-666 Outlook		3356-3357 WWW WWW	9060-9070	CloudOffice	
667-668 Outlook		3358-3359 WWW WWW	9070-9080	CloudOffice	
669-670 Outlook		3360-3361 WWW WWW	9080-9090	CloudOffice	
671-672 Outlook		3362-3363 WWW WWW	9090-9100	CloudOffice	
673-674 Outlook		3364-3365 WWW WWW	9100-9110	CloudOffice	
675-676 Outlook		3366-3367 WWW WWW	9110-9120	CloudOffice	
677-678 Outlook		3368-3369 WWW WWW	9120-9130	CloudOffice	
679-680 Outlook		3370-3371 WWW WWW	9130-9140	CloudOffice	
681-682 Outlook		3372-3373 WWW WWW	9140-9150	CloudOffice	
683-684 Outlook		3374-3375 WWW WWW	9150-9160	CloudOffice	
685-686 Outlook		3376-3377 WWW WWW	9160-9170	CloudOffice	
687-688 Outlook		3378-3379 WWW WWW	9170-9180	CloudOffice	
689-690 Outlook		3380-3381 WWW WWW	9180-9190	CloudOffice	
691-692 Outlook		3382-3383 WWW WWW	9190-9200	CloudOffice	
693-694 Outlook		3384-3385 WWW WWW	9200-9210	CloudOffice	
695-696 Outlook		3386-3387 WWW WWW	9210-9220	CloudOffice	
697-698 Outlook		3388-3389 WWW WWW	9220-9230	CloudOffice	
699-700 Outlook		3390-3391 WWW WWW	9230-9240	CloudOffice	
701-702 Outlook		3392-3393 WWW WWW	9240-9250	CloudOffice	
703-704 Outlook		3394-3395 WWW WWW	9250-9260	CloudOffice	
705-706 Outlook		3396-3397 WWW WWW	9260-9270	CloudOffice	
707-708 Outlook					

Tipos de *sockets* (#1)

- A API socket define três tipos de sockets
 - 1. socket *stream*
 - Interface para o protocolo de transporte TCP
 - 2. socket *datagram*
 - Interface para o protocolo de transporte UDP
 - 3. socket raw



socket raw >>



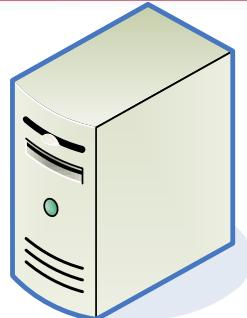
Tipos de *sockets* (#2)

3. Socket *raw*

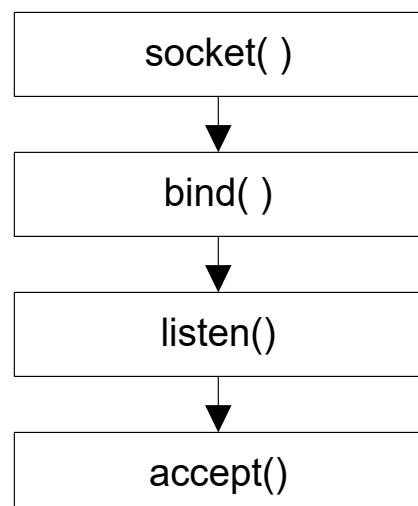
- Interface para o protocolo de rede IP
- Possibilita que aplicações com privilégio de *root*:
 - Capturem o tráfego de rede que passa pelo computador
 - Construam pacotes a partir do zero, incluindo a criação de cabeçalhos de protocolo
- Algumas aplicações maliciosas abusam desta funcionalidade
 - Roubar informações sensíveis, forjar fluxos de rede, contornar os monitores de intrusão de rede e implementar protocolos personalizados de comando e controlo.
- É empregue por aplicações de análise de rede
 - Exemplo: wireshark, nmap, etc.
- Em sistemas Unix e Windows a criação de um socket do tipo *raw* requer privilégios de administração
- Em sistemas MacOS X e FreeBSD deve ser empregue uma biblioteca (e.g., libpcap)

FLUXOGRAMAS TCP & UDP

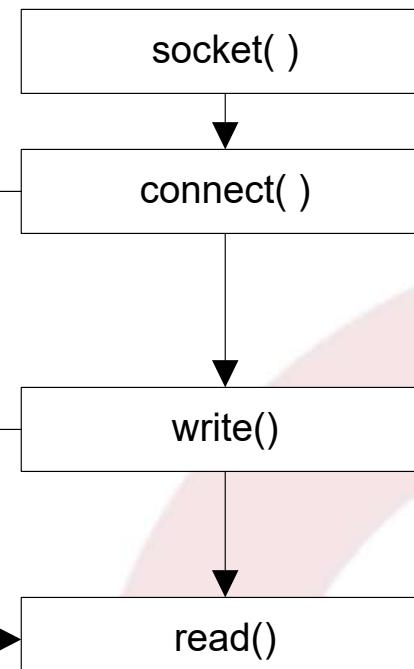
Funções API Socket: TCP



Servidor TCP

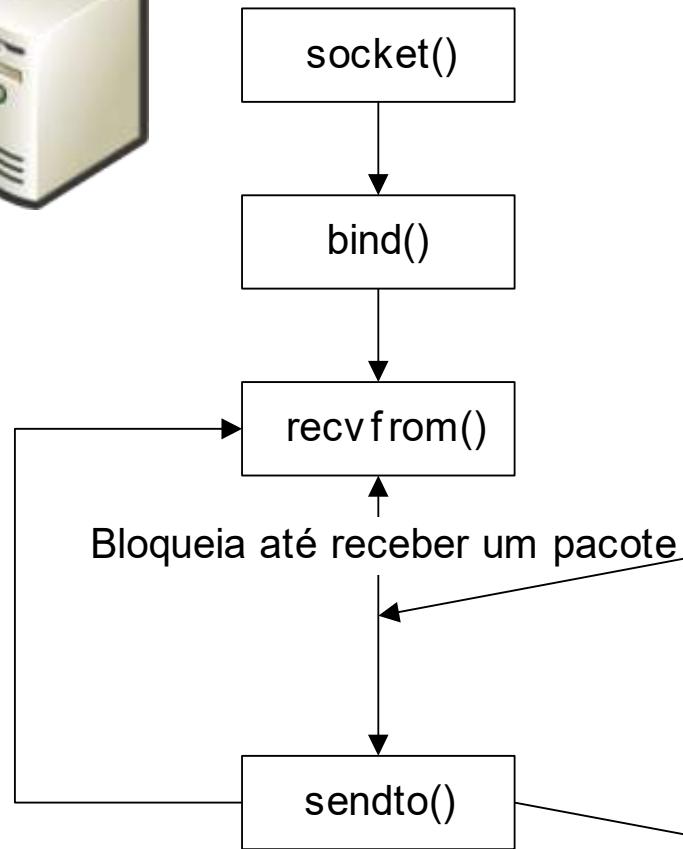
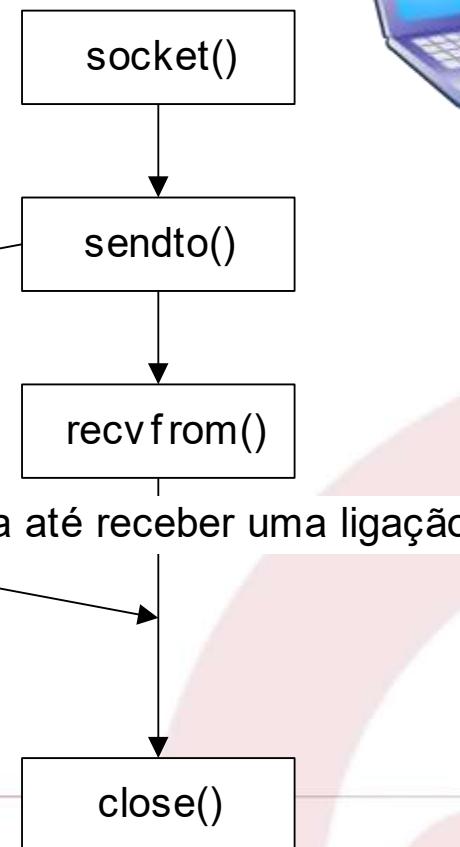


Cliente TCP



ligação estabelecida

Funções API Socket: UDP

**UDP Server****UDP Client**

FUNÇÕES DA API SOCKET: SOCKET()

Funções da API Socket: socket (1)

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

- Criação de um descritor de socket
- Apenas envolve a tabela de descritores do processo da máquina local
- Parâmetros:
 - Domain
 - Define o domínio de comunicação (AF_INET, etc)
 - Type
 - Define o tipo de comunicação (Stream, Datagram, etc)
 - Protocol
 - Define o protocolo específico a utilizar
 - » Na maioria das situações, apenas existe um protocolo por tipo de comunicação
 - » Normalmente passa-se o valor 0 (zero)



stdin
stdout
stderr
x
Socket 1
(...)

**Tabela
descritores**

Funções da API Socket: socket (2)

- `int socket(int domain, int type, int protocol);`
 - Domínios mais comuns
 - AF_UNIX/AF_LOCAL
 - Socket local
 - AF_INET/AF_INET6
 - Socket “Internetwork” IPv4/IPv6
 - Outros (depende da implementação)
 - AF_ISO
 - Socket ISO (OSI)
 - AF_NS
 - Socket XNS (*Xerox Network System*)
 - AF_PACKET, AF_IMPLINK
 - Socket baixo nível
 - ...



Socket (continuação) >>

Funções da API Socket: socket (3)

- `int socket(int domain, int type, int protocol);`
 - Domínios (duas notações para as constantes)
 - AF_xxx vs PF_xxx
 - O prefixo AF é o acrónimo de “Address Family”
 - O prefixo PF é o acrónimo de “Protocol Family”
 - AF_UNIX, AF_LOCAL Local communication
 - AF_INET IPv4 Internet protocol
 - AF_INET6 IPv6 Internet protocol
 - AF_IPX IPX - Novell protocols
 - AF_NETLINK Kernel user interface device
 - AF_X25 ITU-T X.25 / ISO-8208 protocol
 - AF_AX25 Amateur radio AX.25 protocol
 - AF_ATMPVC Access to raw ATM PVCs
 - AF_APPLETALK Appletalk
 - AF_PACKET Low level packet interface
 - Nota: consultar a página de manual: `man socket`

- **int socket(int domain, int type, int protocol);**
 - Tipo de comunicação a utilizar
 - **SOCK_STREAM**
 - Comunicação bidireccional (estabelece ligação, é fiável e sequencial)
 - Exemplo: TCP (em AF_INET ou AF_INET6)
 - **SOCK_DGRAM**
 - Comunicação por datagrams (não estabelece ligação e não é fiável)
 - Exemplo: UDP (em AF_INET ou AF_INET6)
 - **SOCK_RAW**
 - Acesso à camada de rede (socket baixo nível)
 - **SOCK_SEQPACKET**
 - Comunicação bidireccional por datagramas com estabelecimento de ligação (fiável e sequencial)
 - » Não está implementado para o domínio AF_INET
 - » Empregue para protocolos X.25 e AX.25

- **int socket(int domain, int type, int protocol);**
 - Protocolo a utilizar (usualmente a combinação <domínio, tipo> define o protocolo)
 - (**AF_INET, SOCK_STREAM**) => TCP
 - (**AF_INET, SOCK_DGRAM**) => UDP
- Exemplo:

```
int serverfd;  
  
if ( (serverfd=socket(AF_INET, SOCK_STREAM, 0)) == -1 ) {  
    ERROR(-1, "Nao criou o socket");  
}
```

Socket raw >>

- **int socket(int domain, int type, int protocol);**
 - Criação de um socket *raw*
 - Lembrete – *socket raw*
 - Interação direta com a camada IP
 - type: SOCK_RAW
 - protocol: IPPROTO_RAW
- Exemplo:

```
int socket_fd;  
  
if ( (socked_fd=socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) == -1 ) {  
    ERROR(-1, "Nao criou o socket");  
}
```

FUNÇÕES DA API SOCKET: BIND()

Funções da API Socket: bind(1)

```
int bind(int sockfd,  
        const struct sockaddr* my_addr, socklen_t addrLen);
```

- Registo do socket no sistema
 - Associa um endereço ao socket
 - endereço IP local + porto local
 - Os pacotes que o sistema recebe no porto especificado são associados ao processo que registou o socket (processo que efectuou o *bind*)
- Parâmetros
 - **sockfd**
 - Descritor do socket a registrar no sistema
 - **my_addr**
 - Endereço a associar ao socket (contém o porto)
 - **addrlen**
 - Tamanho do endereço

Funções da API Socket: bind(2)

```
int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);
```

- Estrutura genérica de um endereço

```
struct sockaddr {  
    uint8_t      sa_len;  
    sa_family_t  sa_family;  
    char         sa_data[14];  
};
```

- Porquê uma estrutura genérica para o endereço?
 - Existem várias famílias de endereços (AF_INET, AF_INET6, AF_UNIX, ...)
 - Algumas funções da API sockets recebem como parâmetros endereços
 - A API sockets existe antes do ANSI C.
- Solução
 - as funções recebem um ponteiro para uma estrutura genérica de endereços
 - O campo `sa_family` é preenchido com o código correspondente à família de endereço efetivamente colocado na estrutura

ENDEREÇOS NA API SOCKET



IPL

escola superior de tecnologia e gestão
Instituto Politécnico de Leiria

Funções da API Socket: representação de endereços (1)

IPv4

sockaddr_in{}	
length	AF_INET
16-bit port#	
32-bit IPv4 address	
Unused	

fixed length (16 bytes)

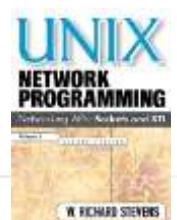
IPv6

sockaddr_in6{}	
length	AF_INET6
16-bit port#	
32-bit flow label	
128-bit IPv6 address	

fixed length (24 bytes)

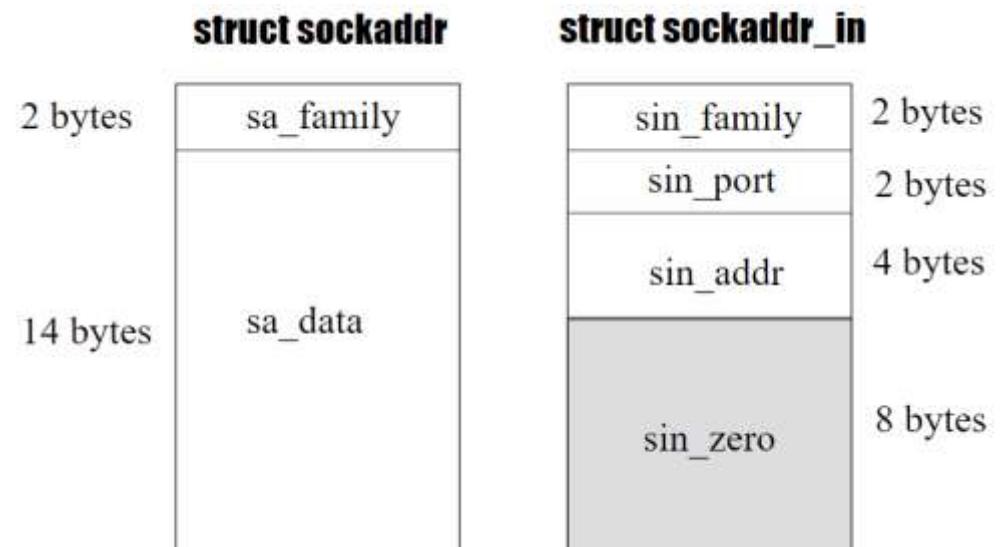
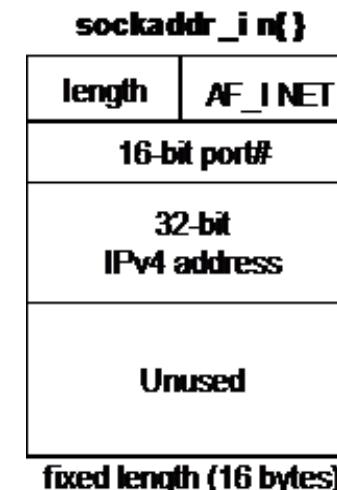
Unix

sockaddr_un{}	
length	AF_LOCAL
pathname (up to 104 bytes)	variable length



- Para cada família de endereços existe uma estrutura específica
 - Para os endereços IPv4 (família AF_INET) existe a seguinte estrutura

```
struct sockaddr_in {
    uint8_t      sin_len;
    sa_family_t  sin_family;
    // número do porto
    in_port_t    sin_port;
    // endereço IP 32 bits
    struct in_addr sin_addr;
    // Padding
    char sin_zero[8];
};
```



Um ponteiro para **struct sockaddr_in** pode ser convertido (cast) para **struct sockaddr**



- Um endereço IP da família AF_INET é representado pela seguinte estrutura

```
struct in_addr {  
    in_addr_t s_addr; // endereço IP (32 bits)  
};
```

- Porquê uma estrutura?

The reason the `sin_addr` member is a structure, and not just an `in_addr_t`, is historical.

Earlier releases (4.2BSD) defined the `in_addr` structure as a union of various structures, to allow access to each of the 4 bytes and to both of the 16-bit values contained within the 32-bit IPv4 address. This was used with class A, B, and C addresses to fetch the appropriate bytes of the address. But with the advent of subnetting and then the disappearance of the various address classes with classless addressing, the need for the union disappeared. Most systems today have done away with the union and just define `in_addr` as a structure with a single `in_addr_t` member. **Fonte:** Addison Wesley : UNIX Network Programming Volume 1, Third Edition:

Funções da API Socket: endereços IPV6

- Endereços IPV6 (família AF_INET6 ou PF_INET6)

```
struct sockaddr_in6 {  
    sa_family_t sin6_family; /* AF_INET6 */  
    in_port_t sin6_port; /* porto */  
    uint32_t sin6_flowinfo; /* info fluxo */  
    struct in6_addr sin6_addr; /* endereço IPv6 */  
    uint32_t sin6_scope_id; /* conjunto  
                            interfaces (escopo) */  
};
```

```
struct in6_addr {  
    uint8_t s6_addr[16]; /* endereço IPv6 */  
};
```

IPv6

sockaddr_in6{}

length	AF_INET6
16-bit port#	
32-bit flow label	
128-bit IPv6 address	
fixed length (24 bytes)	



- O valor de endereços e portos devem estar no **formato de rede**
 - Os campos ***sin_port*** e ***sin_addr*** devem ser especificados no formato de rede
 - Formato de rede = BIG endian
 - RFC 1700, 1994 (<http://tools.ietf.org/html/rfc1700>)
 - «The convention in the documentation of Internet Protocols is to express numbers in decimal and to picture data in "big-endian" order. That is, fields are described left to right, with the most significant octet on the left and the least significant octet on the right.»
 - **Erro**
 - Esquecer a conversão para o formato de rede (e vice-versa)
- 

- Funções de conversão (tradicionais)

```
uint16_t htons(uint16_t);
```

```
uint16_t ntohs(uint16_t);
```

```
uint32_t htonl(uint32_t);
```

```
uint32_t ntohl(uint32_t);
```

- Legenda:

- **n** – ordem da rede (network)
- **h** – ordem do “hospedeiro” (host)
- **s** – inteiro de 16 bits (short)
- **l** – inteiro de 32 bits (long)

```
#ifdef __OPTIMIZE__
/* We can optimize calls to the conversion functions. Either nothing has
   to be done or we are using directly the byte-swapping functions which
   often can be inlined. */
# if __BYTE_ORDER == __BIG_ENDIAN
/* The host byte order is the same as network byte order,
   so these functions are all just identity. */
# define ntohl(x)      __uint32_identity (x)
# define ntohs(x)      __uint16_identity (x)
# define htonl(x)      __uint32_identity (x)
# define htons(x)      __uint16_identity (x)
# else
# if __BYTE_ORDER == __LITTLE_ENDIAN
# define ntohl(x)      __bswap_32 (x)
# define ntohs(x)      __bswap_16 (x)
# define htonl(x)      __bswap_32 (x)
# define htons(x)      __bswap_16 (x)
# endif
# endif
# endif
```

#include <netinet/in.h>



■ Funções de conversão disponíveis na glibc

- Não fazem parte da norma da linguagem C
- Requerem a existência da constante préprocessador `_DEFAULT_SOURCE`

```
#include <endian.h>
```

```
uint16_t htobe16(uint16_t host_16bits);
uint16_t htole16(uint16_t host_16bits);
uint16_t be16toh(uint16_t big_endian_16bits);
uint16_t le16toh(uint16_t little_endian_16bits);

uint32_t htobe32(uint32_t host_32bits);
uint32_t htole32(uint32_t host_32bits);
uint32_t be32toh(uint32_t big_endian_32bits);
uint32_t le32toh(uint32_t little_endian_32bits);

uint64_t htobe64(uint64_t host_64bits);
uint64_t htole64(uint64_t host_64bits);
uint64_t be64toh(uint64_t big_endian_64bits);
uint64_t le64toh(uint64_t little_endian_64bits);
```

Macros de *byteswap*

- Macros de troca de blocos de octetos
 - Devolvem os octetos do parâmetro X por ordem invertida

```
#include <byteswap.h>
bswap_16(x);
bswap_32(x);
bswap_64(x);
```

```
#include <stdio.h>
#include <byteswap.h>
#include <stdint.h>

int main(void)
    uint32_t u32 = 0x11223344;
    printf("u32=0x%X=> 0x%X\n",
           u32, bswap_32(u32));
    return 0;
}
```

u32=0x11223344 ==> 0x44332211

CONVERSÃO DE ENDEREÇOS



- Um endereço IPv4 é representado por um inteiro de 32 bits. Como especificar este valor?

```
int inet_aton(char *in, struct in_addr *out);
```

- Converte uma string no formato **ASCII-dotted-decimal** (“192.168.234.243”) para um endereço IPv4 no formato de rede
- Retorno:
 - 0 Erro; <> 0 OK

```
char* inet_ntoa(struct in_addr *in);
```

- Converte um endereço IPv4 especificado no formato de rede para uma string no formato ASCII-dotted-decimal
- Retorna endereço de string no formato **dotted decimal**
- Função simétrica da função **inet_aton**

Perigos do **inet_ntoa** >>

Ainda sobre inet_ntoa

- **char* inet_ntoa(struct in_addr *in);**
 - Converte um endereço IPv4 especificado no formato de rede para uma string no formato ASCII-dotted-decimal
- Mas de onde vêm o espaço em memória empregue para devolver a string?
 - O que diz o manual? (`man inet_ntoa`)
 - “The string is returned in a statically allocated buffer, which subsequent calls will overwrite”
 - A função não é reentrante, nem thread-safe!
- Versão reentrante

`char* inet_ntoa_r(struct in_addr in, char *buf, socklen_t size);`

inet_ntoa / inet_aton: uso desaconselhado

- Então e os endereços IPv6?
 - funções **inet_pton** e **inet_ntop** (IPv4 e IPv6)
 - As funções são reentrantes
- **int inet_pton(int family, const char *src, void *dst);**
 - Converte uma string no formato ASCII-dotted-decimal (“192.168.234.243” ou “0:0:0:0:0:0:1”) para endereço no formato de rede
 - A família de endereços (AF_INET, AF_INET6) é especificada pelo parâmetro **family**
- **char* inet_ntop(int family, const void *src, char *dst, size_t cnt);**
 - Converte um endereço especificado no formato de rede para uma string no formato ASCII-dotted-decimal
 - Função *inversa* da função **inet_pton**

Sobre endereços IPv6... (1)

- Um endereço IPv6
 - tem 128 bits/16 octetos
 - É escrito com 8 blocos, cada bloco tem 16 bits
 - Cada bloco é representado em HEX com valor entre 0 e 0xFFFF (os zeros à esquerda podem ser omitidos)
 - É permitido que os 4 bytes menos significativos sejam um endereço IPv4 em formato doted-decimal
 - Exemplo 1:2:3:4:5:6:192.193.194.195

Continua >>

- Endereço IPv6
 - Permitido o uso da representação :: (*double colon*) para representar um ou mais blocos com zeros
 - Exemplo 1:2::7:8 → 1:2:0:0:0:0:7:8
 - Apenas é permitido a existência de um ::, caso contrário o endereço IPv6 não é considerado válido
 - Exemplo: ::1:: → errado
 - A representação :: não pode aparecer no endereço IPv4 (4 bytes menos significativos)
 - Endereço de *loopback* (domínio *localhost*)
 - IPv4=127.0.0.1, IPv6=::1

Funções da API Socket: bind(4)

- Determinação do par (endereço, porto) na chamada do **bind**
- **INADDR_ANY** (ou **in6addr_any** no IPv6)
 - Constante que permite que o socket seja registado no porto local para todas as interfaces IP que a máquina possa ter
 - um computador pode ter várias interfaces IP (e.g., várias placas de rede, etc.)

INADDR_ANY



Process specifies		Result
IP address	port	
wildcard	0	kernel chooses IP address and port
wildcard	nonzero	kernel chooses IP address, process specifies port
local IP address	0	process specifies IP address, kernel chooses port
local IP address	nonzero	process specifies IP address and port

Figure 4.5 Result when specifying IP address and/or port number to bind.



- Preenchimento da estrutura de endereço para bind

```
#define PORTO 6080
struct sockaddr_in server_addr;
int serverfd;

...
/* inicia a estrutura com zeros */
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(PORTO);

if (bind(serverfd,
          (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1){
    ERROR(1, "Bind ao porto");
}
```

FUNÇÕES DA API SOCKET: CONNECT()

Funções da API Socket: connect(1)

```
int connect(int sockfd, const struct sockaddr* servaddr, socklen_t addrlen);
```



- Chamada no cliente TCP
- Estabelece uma ligação com o servidor TCP especificado em *servaddr*
 - Parâmetros:
 - sockfd
 - Descritor do socket do cliente
 - servaddr
 - Endereço do servidor (endereço IP + porto)
 - addrlen
 - Tamanho do endereço
 - Retorno:
 - 0 OK; -1 Erro

Funções da API Socket: connect(2)

■ Exemplo

```
#define PORTO 6080

struct sockaddr_in server_addr;
char *IP_servidor = "192.168.234.244";
int clientfd;

...
/* inicia com zeros */
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
if (inet_pton(AF_INET, IP_servidor, &server_addr.sin_addr) <= 0){
    ERROR(1, "Invalid address");
}
server_addr.sin_port = htons(PORTO);
if (connect(clientfd, (struct sockaddr *)&server_addr, sizeof(server_addr))
    == -1){
    ERROR(2, "Cannot connect to the server");
}
```

FUNÇÕES DA API SOCKET: LISTEN()

```
int listen(int sockfd, int backlog);
```

- Função empregue por um servidor TCP para assinalar que está pronto a aceitar ligações
- Permite também definir o tamanho da fila de espera (*backlog*)

- **sockfd** – identificador do socket
- **backlog** - tamanho máximo da fila de espera de pedido de ligações PENDENTES
 - Pedidos de ligações recebidos pelos sistemas operativos, mas ainda não aceites pela aplicação
 - Caso a fila de espera de pedidos de ligações esteja cheia, o servidor não envia nada para o cliente
 - Cliente assume que o pacote se perdeu, e volta a reenviar pedido de ligação
 - O máximo de ligações pendentes é definido pela constante **SOMAXCONN** (<sys/socket.h>)

```
int listen(int sockfd, int backlog);
```

- Retorno:

- 0 OK; -1 Erro

```
if (listen(serverfd, 5) == -1){  
    ERROR(1, "Nao foi possivel efetuar o listen");  
}
```

```
int listen(int sockfd, int backlog);
```

- O que sucede se a função `listen` é chamada com um socket que não foi registado? (i.e., não foi efetuado o `bind` ao socket)
 - O sistema efetua um *bind* automático
 - Interface **INADDR_ANY** (IPv4) ou **in6addr_any** (IPv6)
 - Porto é selecionado automaticamente
 - Como determinar o porto que foi atribuído?
 - `int getsockname(int sockfd,
 struct sockaddr *addr,
 socklen_t *addrlen);`

FUNÇÕES DA API SOCKET: ACCEPT()



IPL

escola superior de tecnologia e gestão
Instituto Politécnico de Leiria

Funções da API Socket: accept(#1)

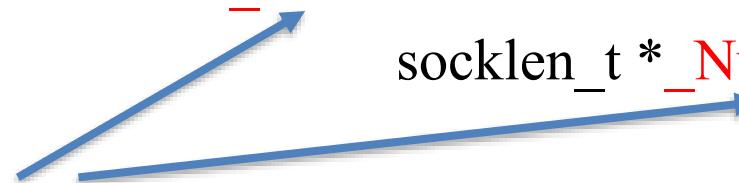
`int accept(int sockfd, struct sockaddr* cliaddr, socklen_t* addrlen);`

- Função utilizada por um servidor TCP para aguardar um pedido de ligação
 - Fica bloqueada até que haja um pedido de ligação
 - Cria um novo socket quando ocorre um pedido de ligação
- Parâmetros:
 - sockfd
 - Descritor do socket do servidor
 - cliaddr
 - Endereço do cliente com o qual acabou de estabelecer uma ligação (endereço IP + porto)
 - addrallen
 - Tamanho do endereço. É necessário passar um ponteiro (passagem por referência)
- Retorno:
 - Descritor do socket do cliente (> 0) OK;
 - -1 Erro

Mais sobre o protótipo do accept (**restrict** and **_Nullable**)>>

accept - protótipo “estendido”

```
int accept(int sockfd, struct sockaddr *_Nullable restrict addr,  
          socklen_t *_Nullable restrict addrlen);
```

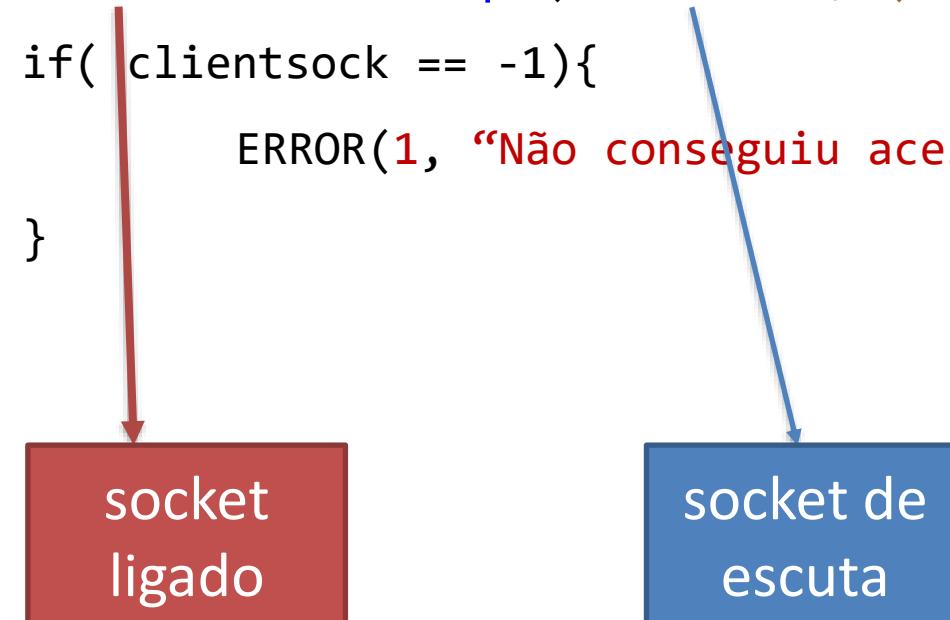


- O qualificador “_Nullable” significa que o parâmetro associado pode ser NULL. É o caso dos parâmetros **addr** e **addrlen**.
- A palavra-chave “**restrict**” é uma indicação do programador ao compilador que o acesso ao endereço de memória apontado pelo ponteiro apenas pode ser feito pelo ponteiro que está a ser indicado como parâmetro.
 - A indicação **restrict** permite que o compilador possa otimizar o código sem receio que existam “alias” para o endereço de memória

- Exemplo

```
struct sockaddr_in client_addr;  
int serversock, clientsock;  
int len = sizeof(client_addr);  
...  
clientsock = accept(serversock, (struct sockaddr *)&client_addr, &len);
```

```
if( clientsock == -1){  
    ERROR(1, "Não conseguiu aceitar a ligação");  
}
```



A diagram illustrating the effect of the accept function. A red vertical arrow points downwards from the 'clientsock' variable in the code to a red rectangular box labeled 'socket ligado' (connected socket). A blue diagonal arrow points from the 'serversock' variable to a blue rectangular box labeled 'socket de escuta' (listening socket).

socket ligado

socket de
escuta

■ Exemplo

```
struct sockaddr_in client_addr;  
  
int serversock, clientsock;  
  
int len = sizeof(client_addr);  
  
...  
  
clientsock = accept(serversock, (struct sockaddr *)&client_addr, &len);  
if( clientsock == -1){  
    ERROR(1, "Não conseguiu aceitar a ligação");  
}
```

- NOTA: o novo *socket clientsock* (socket ligado) usa o mesmo porto TCP local que está a ser empregue como porto de escuta pelo socket *serversock* (socket de escuta)

Funções da API Socket: accept(#5)

```
struct sockaddr_in client_addr;
int serverfd, clientfd;
int len = sizeof(client_addr);
/* Para ser mais correcto, a lista de
endereços proibidos deveria ser
construída dinamicamente a partir de
um ficheiro */
char * proibidos = {"192.168.5.", "192.168.234.22", 0};

...
if ((clientfd = accept(serverfd, (struct sockaddr *)&client_addr, &len)) == -1)
    ERROR(1, "Não conseguiu aceitar a ligação");
if ( eProibido (proibidos, inet_ntoa(client_addr.sin_addr)) ) {
    printf("Ligaçao recusada ao IP: %\n", inet_ntoa(client_addr.sin_addr));
    close(clientfd);
} ...
```

```
int eProibido(char * lista[], char *ip_cliente) {
    int i = 0;
    while (lista[i])
        if (strstr(lista[i++], ip_cliente) != NULL)
            return 1;
    return 0;
}
```

O mais correto será utilizar a função
inet_ntop (mais genérica, suporta IPV6)

FUNÇÕES DA API SOCKET: ENVIO/RECEÇÃO DE DADOS



■ Funções orientadas à ligação

`ssize_t read(int fd, void* buffer, size_t nbytes);`

- Lê do descritor *fd*, *nbytes* para a zona de memória apontada por *buffer*
- Retorno:
 - OK: número de bytes lidos
 - Erro: **-1**

`ssize_t write(int fd, void* buffer, size_t nbytes);`

- Escreve *nbytes* do conteúdo da zona de memória apontada por *buffer* para o descritor *fd*
- Retorno:
 - OK: número de bytes escritos
 - Erro: **-1**



■ Funções orientadas à ligação

`ssize_t recv(int s, const void* buf, size_t len, int flags);`

- Lê do socket *s* *len* bytes para a zona de memória apontada por *buf*
- Retorno:
 - OK: número de bytes recebidos;
 - 0: ligação fechada do outro lado
 - Erro: *-1*

`ssize_t send(int s, const void* msg, size_t len, int flags);`

- Escreve *len* bytes do conteúdo da zona de memória apontada por *msg* para o socket *s*
- Retorno:
 - OK: número de bytes enviados;
 - Erro: *-1*

flags: configura comportamento especiais da função



■ Funções não orientadas à ligação

```
ssize_t recvfrom(int s, const void* buf, size_t len,  
int flags, struct sockaddr* from, socklen_t* fromlen);
```

- Lê do socket *s* *len* bytes para a zona de memória apontada por *buf*
- Se *from* <> NULL e *s* for um socket não orientado à ligação então:
 - *from* é preenchido com o endereço origem da mensagem
 - *fromlen* contém o tamanho do endereço
- Retorno
 - OK: número de bytes recebidos
 - Erro: -1



■ Funções não orientadas à ligação

```
ssize_t sendto (int s, const void* msg, size_t len,  
    int flags, const struct sockaddr* to, socklen_t tolen);
```

- Escreve *len* bytes do conteúdo da zona de memória apontada por *msg* para o socket *s*
- Se *to* <> NULL e *s* for um socket não orientado à ligação então a mensagem é enviada para o destinatário com o endereço *to*
- Retorno
 - OK: número de bytes enviados
 - Erro: -1



Funções da API Socket: escrita/leitura (5)

```
ssize_t recv(int s, const void* buf, size_t len, int flags);  
ssize_t recvfrom(int s, const void* buf, size_t len,  
int flags, struct sockaddr* from, socklen_t* fromlen);
```

■ Flags

- 0 – Nenhuma opção
- MSG_OOB – mensagem urgentes (mensagens “out of band” apenas para SOCK_STREAM)
- MSG_PEEK – acesso aos dados sem que sejam retirados do socket
- MSG_NOSIGNAL – desativa “sigpipe” quando a outra extremidade comunicante desaparece
- MSG_WAITALL – bloqueia até a mensagem ser totalmente recebida

...



```
ssize_t send(int s, const void* msg, size_t len, int flags);  
ssize_t sendto (int s, const void* msg, size_t len, int flags,  
               const struct sockaddr* to, socklen_t tolen);
```

- Flags

- 0 – Nenhuma opção
- **MSG_OOB** – mensagem urgentes (mensagens “out of band” apenas para **SOCK_STREAM**)
- **MSG_DONTWAIT** – escrita não-bloqueante
- **MSG_NOSIGNAL** – desativa “sigpipe” quando a outra extremidade comunicante desaparece

....

FUNÇÕES DA API SOCKET: CLOSE()

Funções da API Socket: close

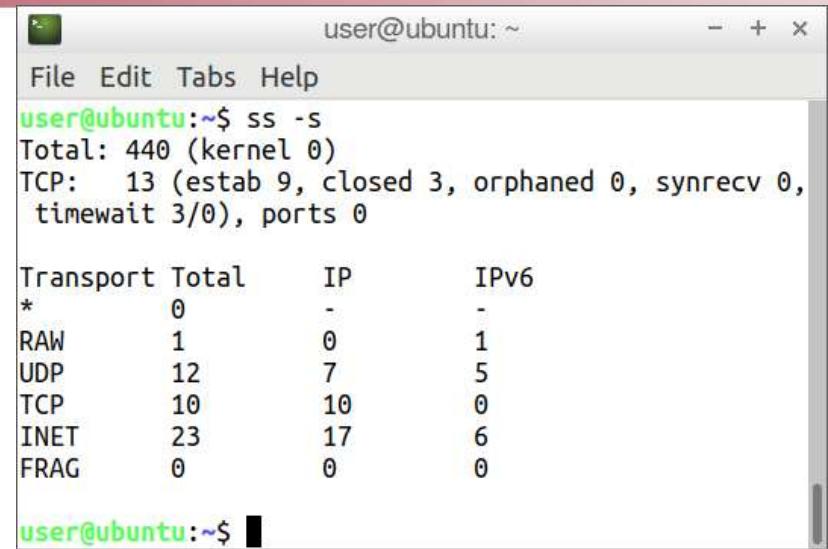
```
int close(int fd);
```



- Faz o descritor *fd*
 - Liberta os recursos ocupados pelo descritor *fd*
- O socket deve ser fechado quando deixa de ser preciso!
 - No que respeita ao S.O., um socket é um descritor que ocupa uma posição na tabela de descritores abertos do processo
 - Cada processo tem um limite para o número de descritores abertos
 - Por exemplo, 1024
- NOTA
 - Quando não é explicitamente fechado, no término do processo o descritor é libertado

Utilitário ss (linux)

- **ss:** informação sobre sockets
- **ss -s**
 - Estatísticas
- **ss -l**
 - Lista portos à escuta
- **ss -t -a**
 - Lista todos os sockets TCP
- **ss -u -a**
 - Lista todos os sockets UDP
- **ss -tp**
 - Opção p: lista PID e executável do processo detentor do socket
- **ss -ie**
 - Mostra informação estendida (estado interno, etc.)



```

user@ubuntu:~$ ss -s
Total: 440 (kernel 0)
TCP:   13 (estab 9, closed 3, orphaned 0, synrecv 0,
       timewait 3/0), ports 0

Transport Total      IP          IPv6
*        0          -          -
RAW     1          0          1
UDP    12          7          5
TCP    10         10          0
INET   23         17          6
FRAG   0          0          0

user@ubuntu:~$ █

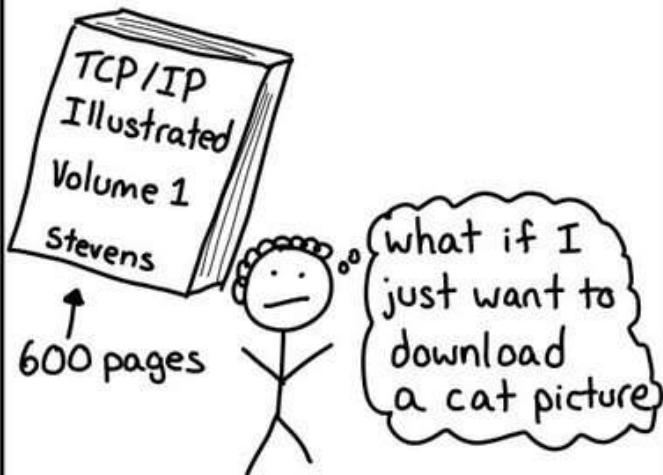
```

- **ss -m**
 - Memória em uso
- **ss -tuln**
 - Mostra todos os portos TCP/UDP em “escuta”
- **ss -tan**
 - Lista todas as ligações TCP com a indicação da entidade remota

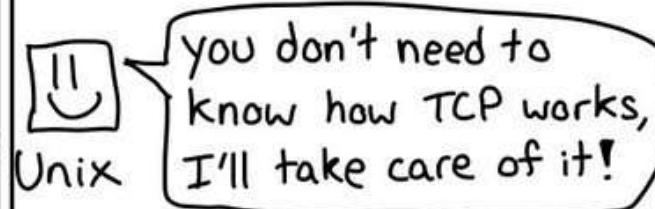
Sockets

drawings.jvns.ca

networking protocols
are complicated



Unix systems have
an API called the
"socket API" that
makes it easier to make
network connections
(Windows too! 😊)



Every HTTP library uses
sockets under the hood

\$ curl awesome.com
Python: requests.get("yay.us")



AF-INET?
What's that?

AF-INET means basically
"internet socket": it lets you
connect to other computers
on the internet using their
IP address.

The main alternative is
AF-UNIX ("unix domain socket")
for connecting to programs
on the same computer

here's what getting
a cat picture with the
socket API looks like:

- ① Create a socket
`fd = socket(AF_INET, SOCK_STREAM ...)`
- ② Connect to an IP/port
`connect(fd, 12.13.14.15:80)`
- ③ Make a request
`write(fd, "GET /cat.png HTTP/1.1 ...")`
- ④ Read the response
`cat-picture = read(fd ...)`

3 kinds of internet
(AF-INET) sockets:

SOCK_STREAM = TCP
curl uses this

SOCK_DGRAM = UDP
dig (DNS) uses this

SOCK_RAW = just let me
send IP packets
I will implement
my own protocol
ping uses this

Bibliografia

- “Beej's Guide to Network Programming - Using Internet Sockets”, Brian “Beej Jorgensen” Hall, 2025
(<http://beej.us/guide/bgnet/>)
- “UNIX Network Programming”, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998, ISBN 0-13-490012-X.
(<http://www.kohala.com/start/unpv12e.html>)
- “Basic Socket Interface Extensions for IPv6”, RFC 3493, Fev. 2003
(<https://www.ietf.org/rfc/rfc3493.txt>)
- man 7 ip
- man 7 ipv6
- man 7 socket

The screenshot shows a terminal window titled "Linux Programmer's Manual" with the command "man ip". The man page for the ip(7) socket type is displayed. It includes sections for NAME, SYNOPSIS, and DESCRIPTION. The SYNOPSIS section shows code snippets for creating TCP, UDP, and RAW sockets using the socket() function. The DESCRIPTION section provides details about the Internet Protocol version 4 implementation in Linux.

```
NAME
    ip - Linux IPv4 protocol implementation

SYNOPSIS
    #include <sys/socket.h>
    #include <netinet/in.h>
    #include <netinet/ip.h> /* superset of previous */

    tcp_socket = socket(AF_INET, SOCK_STREAM, 0);
    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
    raw_socket = socket(AF_INET, SOCK_RAW, protocol);

DESCRIPTION
    Linux implements the Internet Protocol, version 4, described in RFC 791
    and RFC 1122. ip contains a level 2 multicasting implementation conforming
    to RFC 1112. It also contains an IP router including a packet filter.

    The programming interface is BSD-sockets compatible. For more information
    on sockets, see socket(7).

Manual page ip(7) lline 1 (press h for Help or q to quit)
```

