

Sockets (BSD)

Patricio Domingues
(c) Patricio Domingues, Vitor Carreira



A word cloud graphic featuring various programming and system-related terms. The most prominent word is 'Programação Avançada' in large, bold, dark red letters. Other visible words include 'sockets', 'lock/unlock', '#define', 'malloc', 'doxygen', '(c) Patricio Domingues', '#include', 'for', 'char *ptr:', 'tree', 'ponteiro', 'ciclo', 'gcc', 'linked list', 'mutex', 'IPL++', 'gdb', and 'threads'. The words are arranged in a horizontal, slightly curved layout, with varying colors and sizes.

SOCKETS

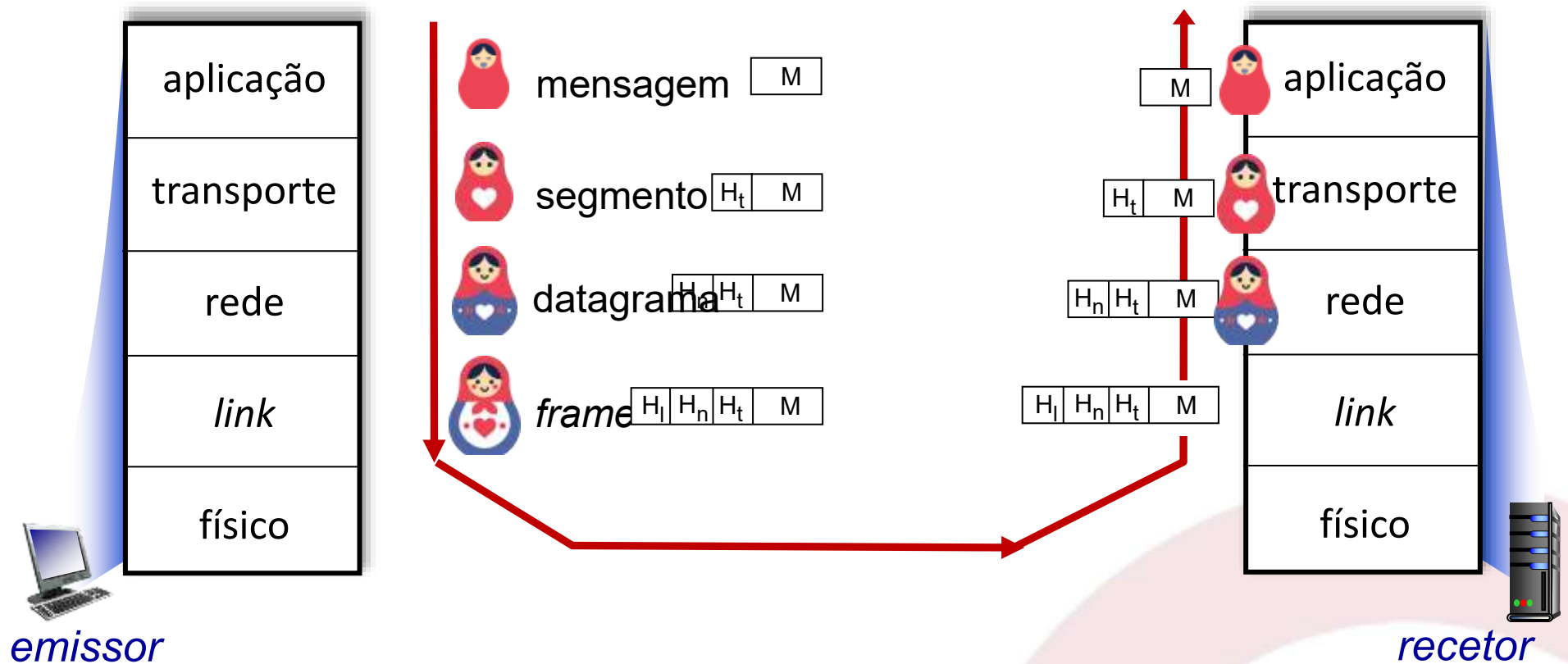




IPL

escola superior de tecnologia e gestão
instituto politécnico de leiria

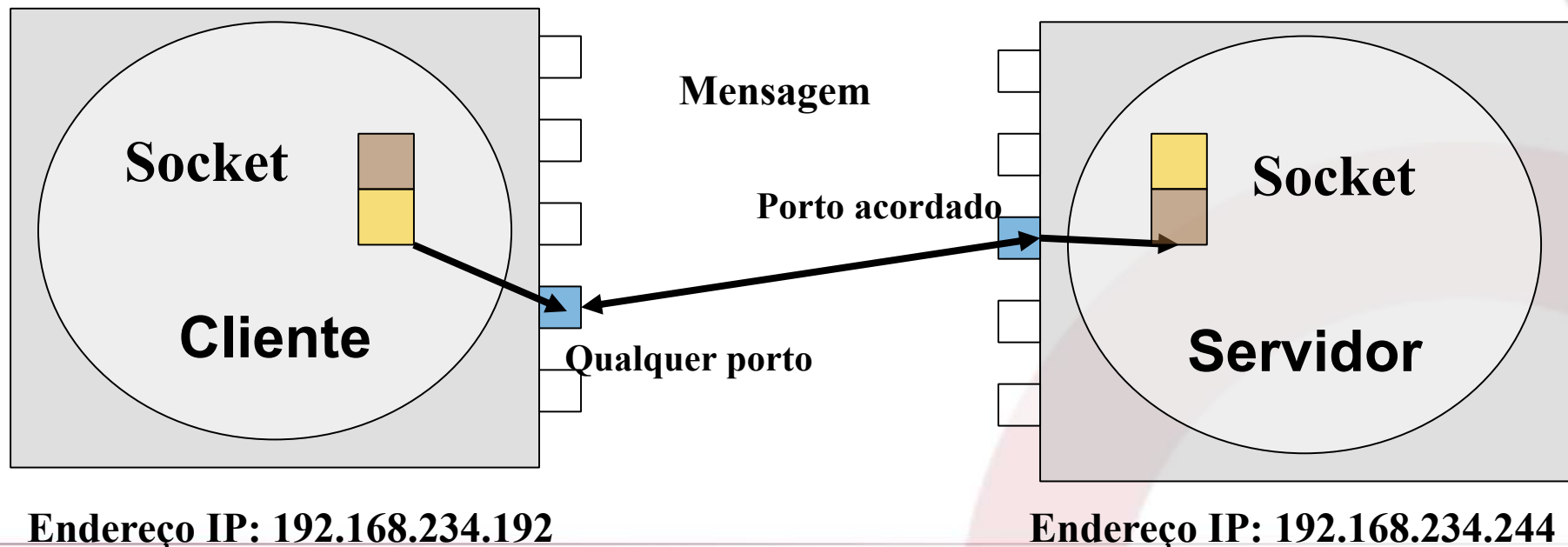
Serviços, camadas e encapsulamento



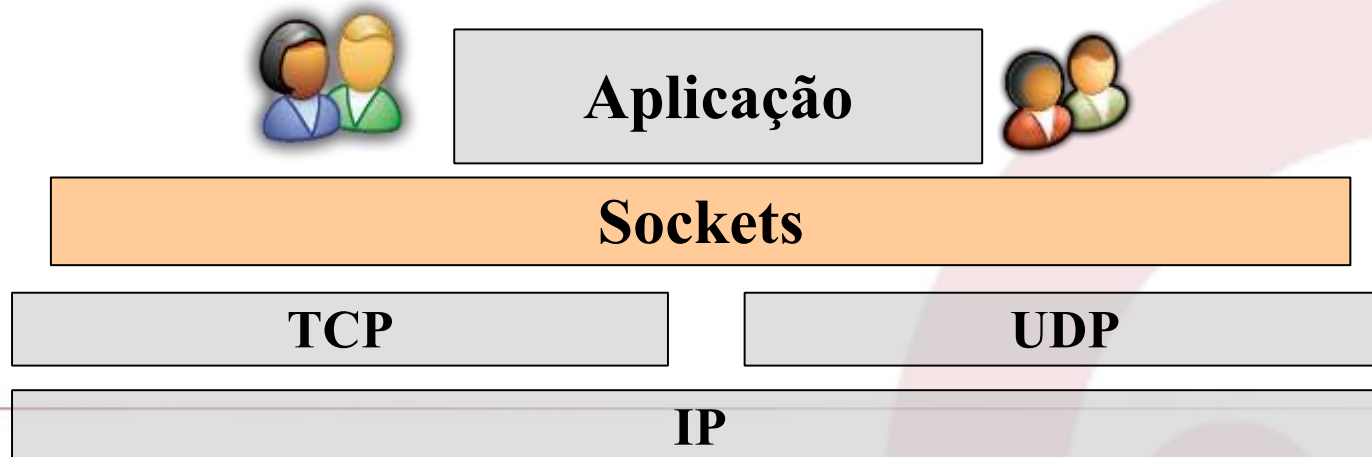
Adaptado de:
Computer Networking: A Top-Down Approach
8th edition, Jim Kurose, Keith Ross, Pearson, 2020

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



- 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)
- 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



Wall socket



- 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
 - ...

- 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)



■ Porto

- Identificador numérico (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

COMMON PORTS

port.kitware.com

TCP/UDP Port Numbers

7 Echo	814 FTP	8145 RDP	8881 XMPP	Unicom
11 Charge	844-847 DHCP	8887 Symantec	8870	Quotient
20-21 FTP	860 unircat	8889 Microsoft	7212	Quotient
22 SSH	902	8894	1444-1449	Unicom
23 Telnet	907 SAT	8894	8894	Unicom
25 SMTP	910 Hadoop	8894	8894	Unicom
42 Microsoft Application	909 Microsoft SCOM	8894	8894	Unicom
43 WHOIS	933 Internet Sharing	8894	8894	Unicom
44 TACACS	934	8894	8894	Unicom
53 DNS	935	8894	8894	Unicom
54-60 Microsoft	936-940	8894	8894	Unicom
66 TFTP	941 Exchange	8894	8894	Unicom
70 Quake	942	8894	8894	Unicom
71 Finger	943	8894	8894	Unicom
80 HTTP	944	8894	8894	Unicom
88 Veritas	945	8894	8894	Unicom
90 MS Exchange	946	8894	8894	Unicom
91-94	947-950	8894	8894	Unicom
113 netbios	1025	8894	8894	Unicom
119 NTP-Client	1026-1028	8894	8894	Unicom
123 NTP	1029	8894	8894	Unicom
135 Microsoft RPC	1030	8894	8894	Unicom
137-139 netbios	1031	8894	8894	Unicom
142-143	1032	8894	8894	Unicom
177 CIFS	1033	8894	8894	Unicom
179 BGP	1034	8894	8894	Unicom
800 AppleTalk	1035-1036	8894	8894	Unicom
204 POP	1037	8894	8894	Unicom
314 POP	1038	8894	8894	Unicom
361-362 MS Exchange	1039	8894	8894	Unicom
380 LDAP	1040	8894	8894	Unicom
411-412	1041	8894	8894	Unicom
443 HTTP	1042	8894	8894	Unicom
445 Microsoft FS	1043	8894	8894	Unicom
464	1044	8894	8894	Unicom
480	1045	8894	8894	Unicom
497	1046	8894	8894	Unicom
500	1047	8894	8894	Unicom
512 net	1048	8894	8894	Unicom
513 BGP	1049	8894	8894	Unicom
514 net	1050	8894	8894	Unicom
515 LDAP	1051	8894	8894	Unicom
516 net	1052	8894	8894	Unicom
520	1053	8894	8894	Unicom
521	1054	8894	8894	Unicom
522	1055	8894	8894	Unicom
523	1056	8894	8894	Unicom
524	1057	8894	8894	Unicom
525	1058	8894	8894	Unicom
526	1059	8894	8894	Unicom
527	1060	8894	8894	Unicom
528	1061	8894	8894	Unicom
529	1062	8894	8894	Unicom
530	1063	8894	8894	Unicom
531	1064	8894	8894	Unicom
532	1065	8894	8894	Unicom
533	1066	8894	8894	Unicom
534	1067	8894	8894	Unicom
535	1068	8894	8894	Unicom
536	1069	8894	8894	Unicom
537	1070	8894	8894	Unicom
538	1071	8894	8894	Unicom
539	1072	8894	8894	Unicom
540	1073	8894	8894	Unicom
541	1074	8894	8894	Unicom
542	1075	8894	8894	Unicom
543	1076	8894	8894	Unicom
544	1077	8894	8894	Unicom
545	1078	8894	8894	Unicom
546	1079	8894	8894	Unicom
547	1080	8894	8894	Unicom
548	1081	8894	8894	Unicom
549	1082	8894	8894	Unicom
550	1083	8894	8894	Unicom
551	1084	8894	8894	Unicom
552	1085	8894	8894	Unicom
553	1086	8894	8894	Unicom
554	1087	8894	8894	Unicom
555	1088	8894	8894	Unicom
556	1089	8894	8894	Unicom
557	1090	8894	8894	Unicom
558	1091	8894	8894	Unicom
559	1092	8894	8894	Unicom
560	1093	8894	8894	Unicom
561	1094	8894	8894	Unicom
562	1095	8894	8894	Unicom
563	1096	8894	8894	Unicom
564	1097	8894	8894	Unicom
565	1098	8894	8894	Unicom
566	1099	8894	8894	Unicom
567	1100	8894	8894	Unicom
568	1101	8894	8894	Unicom
569	1102	8894	8894	Unicom
570	1103	8894	8894	Unicom
571	1104	8894	8894	Unicom
572	1105	8894	8894	Unicom
573	1106	8894	8894	Unicom
574	1107	8894	8894	Unicom
575	1108	8894	8894	Unicom
576	1109	8894	8894	Unicom
577	1110	8894	8894	Unicom
578	1111	8894	8894	Unicom
579	1112	8894	8894	Unicom
580	1113	8894	8894	Unicom
581	1114	8894	8894	Unicom
582	1115	8894	8894	Unicom
583	1116	8894	8894	Unicom
584	1117	8894	8894	Unicom
585	1118	8894	8894	Unicom
586	1119	8894	8894	Unicom
587	1120	8894	8894	Unicom
588	1121	8894	8894	Unicom
589	1122	8894	8894	Unicom
590	1123	8894	8894	Unicom
591	1124	8894	8894	Unicom
592	1125	8894	8894	Unicom
593	1126	8894	8894	Unicom
594	1127	8894	8894	Unicom
595	1128	8894	8894	Unicom
596	1129	8894	8894	Unicom
597	1130	8894	8894	Unicom
598	1131	8894	8894	Unicom
599	1132	8894	8894	Unicom
600	1133	8894	8894	Unicom
601	1134	8894	8894	Unicom
602	1135	8894	8894	Unicom
603	1136	8894	8894	Unicom
604	1137	8894	8894	Unicom
605	1138	8894	8894	Unicom
606	1139	8894	8894	Unicom
607	1140	8894	8894	Unicom
608	1141	8894	8894	Unicom
609	1142	8894	8894	Unicom
610	1143	8894	8894	Unicom
611	1144	8894	8894	Unicom
612	1145	8894	8894	Unicom
613	1146	8894	8894	Unicom
614	1147	8894	8894	Unicom
615	1148	8894	8894	Unicom
616	1149	8894	8894	Unicom
617	1150	8894	8894	Unicom
618	1151	8894	8894	Unicom
619	1152	8894	8894	Unicom
620	1153	8894	8894	Unicom
621	1154	8894	8894	Unicom
622	1155	8894	8894	Unicom
623	1156	8894	8894	Unicom
624	1157	8894	8894	Unicom
625	1158	8894	8894	Unicom
626	1159	8894	8894	Unicom
627	1160	8894	8894	Unicom
628	1161	8894	8894	Unicom
629	1162	8894	8894	Unicom
630	1163	8894	8894	Unicom
631	1164	8894	8894	Unicom
632	1165	8894	8894	Unicom
633	1166	8894	8894	Unicom
634	1167	8894	8894	Unicom
635	1168	8894	8894	Unicom
636	1169	8894	8894	Unicom
637	1170	8894	8894	Unicom
638	1171	8894	8894	Unicom
639	1172	8894	8894	Unicom
640	1173	8894	8894	Unicom
641	1174	8894	8894	Unicom
642	1175	8894	8894	Unicom
643	1176	8894	8894	Unicom
644	1177	8894	8894	Unicom
645	1178	8894	8894	Unicom
646	1179	8894	8894	Unicom
647	1180	8894	8894	Unicom
648	1181	8894	8894	Unicom
649	1182	8894	8894	Unicom
650	1183	8894	8894	Unicom
651	1184	8894	8894	Unicom
652	1185	8894	8894	Unicom
653	1186	8894	8894	Unicom
654	1187	8894	8894	Unicom
655	1188	8894	8894	Unicom
656	1189	8894	8894	Unicom
657	1190	8894	8894	Unicom
658	1191	8894	8894	Unicom
659	1192	8894	8894	Unicom
660	1193	8894	8894	Unicom
661	1194	8894	8894	Unicom
662	1195	8894	8894	Unicom
663	1196	8894	8894	Unicom
664	1197	8894	8894	Unicom
665	1198	8894	8894	Unicom
666	1199	8894	8894	Unicom
667	1200	8894	8894	Unicom
668	1201	8894	8894	Unicom
669	1202	8894	8894	Unicom
670	1203	8894	8894	Unicom
671	1204	8894	8894	Unicom
672	1205	8894	8894	Unicom
673	1206	8894	8894	Unicom
674	1207	8894	8894	Unicom
675	1208	8894	8894	Unicom
676	1209	8894	8894	Unicom
677	1210	8894	8894	Unicom
678	1211	8894	8894	Unicom
679	1212	8894	8894	Unicom
680	1213	8894	8894	Unicom
681	1214	8894	8894	Unicom
682	1215	8894	8894	Unicom
683	1216	8894	8894	Unicom
684	1217	8894	8894	Unicom
685	1218	8894	8894	Unicom
686	1219	8894	8894	Unicom
687	1220	8894	8894	Unicom
688	1221	8894	8894	Unicom
689	1222	8894	8894	Unicom
690	1223	8894	8894	Unicom
691	1224	8894	8894	Unicom
692	1225	8894	8894	Unicom
693	1226	8894	8894	Unicom
694	1227	8894	8894	Unicom
695	1228	8894	8894	Unicom
696	1229	8894	8894	Unicom
697	1230	8894	8894	Unicom
698	1231	8894	8894	Unicom
699	1232	8894	8894	Unicom
700	1233	8894	8894	Unicom
701	1234	8894	8894	Unicom
702	1235	8894	8894	Unicom
703	1236	8894	8894	Unicom
704	1237	8894	8894	Unicom
705	1238	8894	8894	Unicom
706	1239	8894	8894	Unicom
707	1240	8894	8894	Unicom
708	1241	8894	8894	Unicom
709	1242	8894	8894	Unicom
710	1243	8894	8894	Unicom
711	1244	8894	8894	Unicom
712	1245	8894	8894	Unicom
713	1246	8894	8894	Unicom
714	1247	8894	8894	Unicom
715	1248	8894	8894	Unicom
716	1249	8894	8894	Unicom
717	1250	8894	8894	Unicom
718	1251	8894	8894	Unicom
719	1252	8894	8894	Unicom
720	1253	8894	8894	Unicom
721	1254	8894	8894	Unicom
722	1255	8894	8894	Unicom
723	1256	8894	8894	Unicom
724	1257	8894	8894	Unicom
725	1258	8		

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* >>

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)

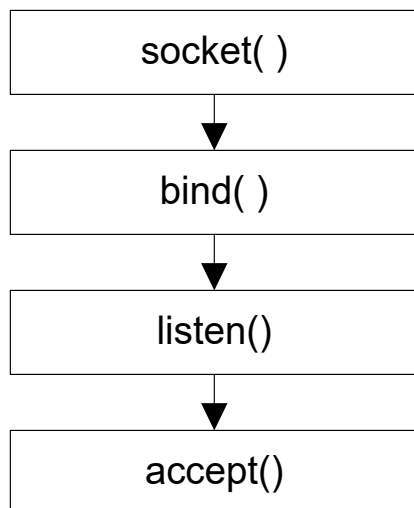
FLUXOGRAMAS TCP & UDP



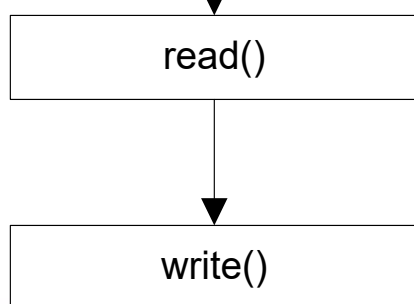
Funções API Socket: TCP



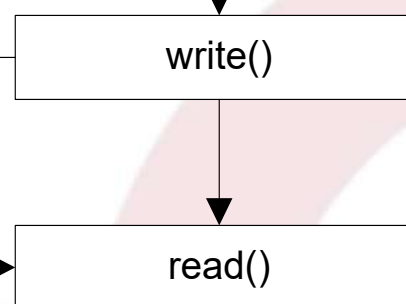
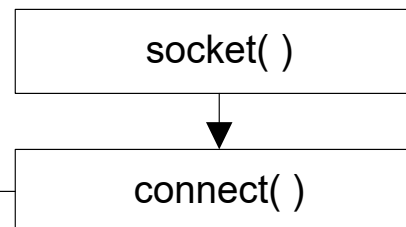
Servidor TCP



ligação estabelecida



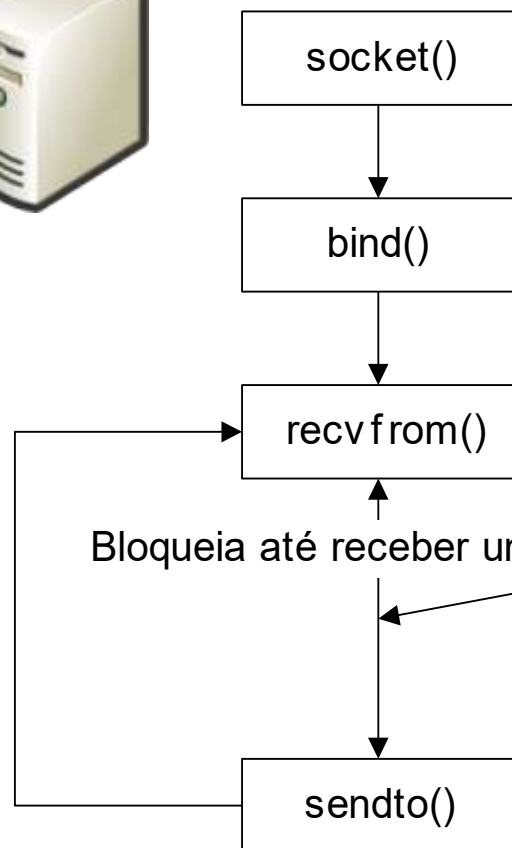
Cliente TCP



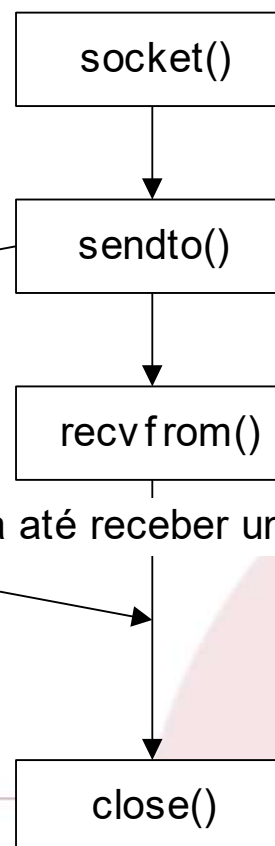
Funções API Socket: UDP



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**

- `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_IMPLNK`
 - Socket baixo nível
 - ...

- `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 ( (socket_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 registar 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





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)	

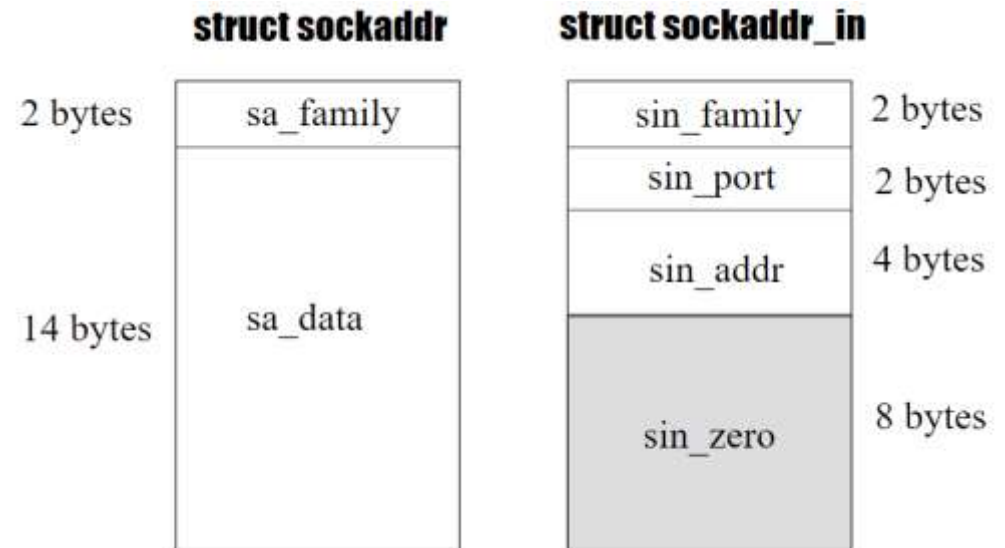
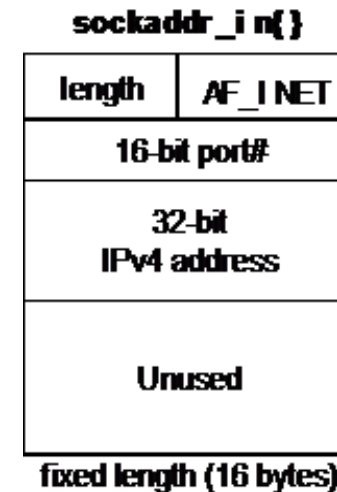
variable length





- 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:*

IPv6

sockaddr_in6{}

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

fixed length (24 bytes)

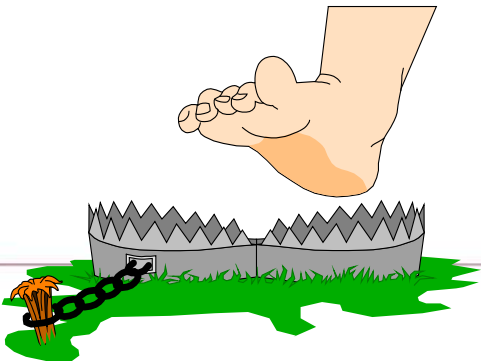
- Endereços IPV6 (família AF_INET6 ou PF_INET6)

```
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 */
};
```



- 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 (**n**etwork)
- **h** – ordem do “hospedeiro” (**h**ost)
- **s** – inteiro de 16 bits (**s**hort)
- **l** – inteiro de 32 bits (**l**ong)

```
#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 “0:0:0:0:0:0:0:1”) 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**

- 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 o `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

■ 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()



Funções da API Socket: listen (#1)

```
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()



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)
 - *addrlen*
 - 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");  
}
```



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("Ligação 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* \neq 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* \neq **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()



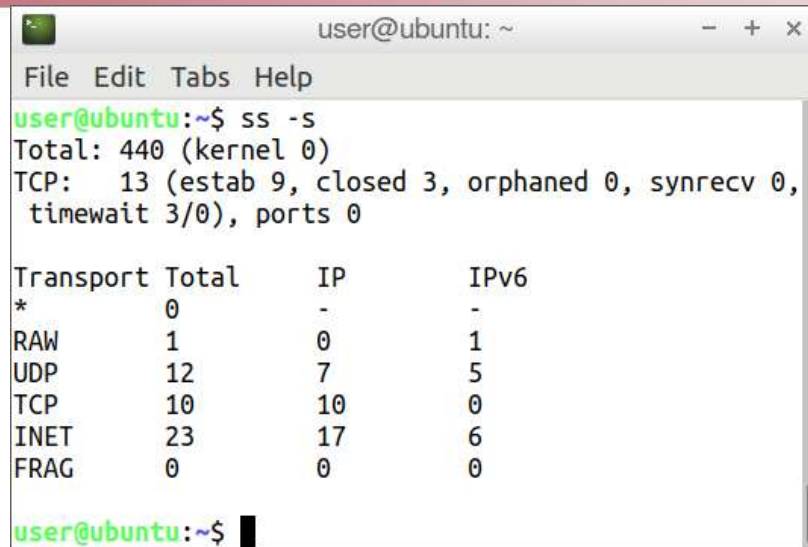
```
int close(int fd);
```



- Fecha 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: ~  
File Edit Tabs Help  
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:~$
```

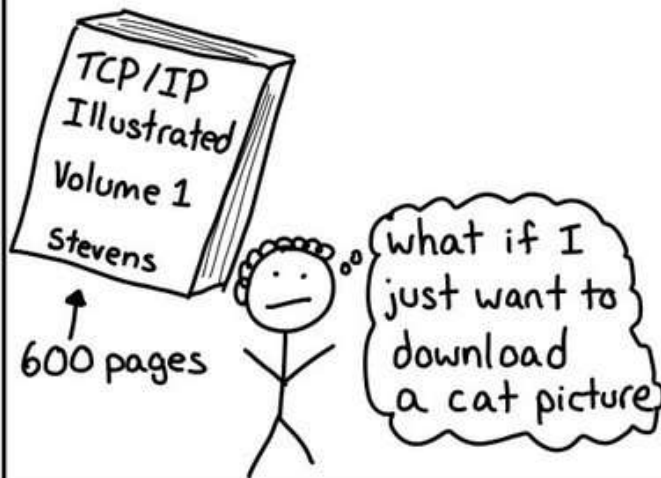
- ss -m
 - Memória em uso
- ss -tuln
 - Mostra todos os portos TCP/UDP em “escuta”
- ss -tan
 - Lista todas as ligações TCP com a indicação da entidade remota

JULIA EVANS
@b0rk

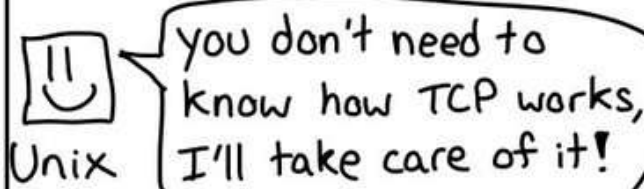
sockets

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! ☺)



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, ...)`

Every HTTP library uses
sockets under the hood

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

↑ SOCKETS



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

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
↑
ping uses this

send IP packets
I will implement
my own protocol

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, Feb. 2003
(<https://www.ietf.org/rfc/rfc3493.txt>)
- man 7 ip
- man 7 ipv6
- man 7 socket



```
man 7 ip
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 793 and RFC 1122. ip contains a level 2 multicasting implementation conforming to RFC 1112. It also contains an IP router including a socket filter.

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

