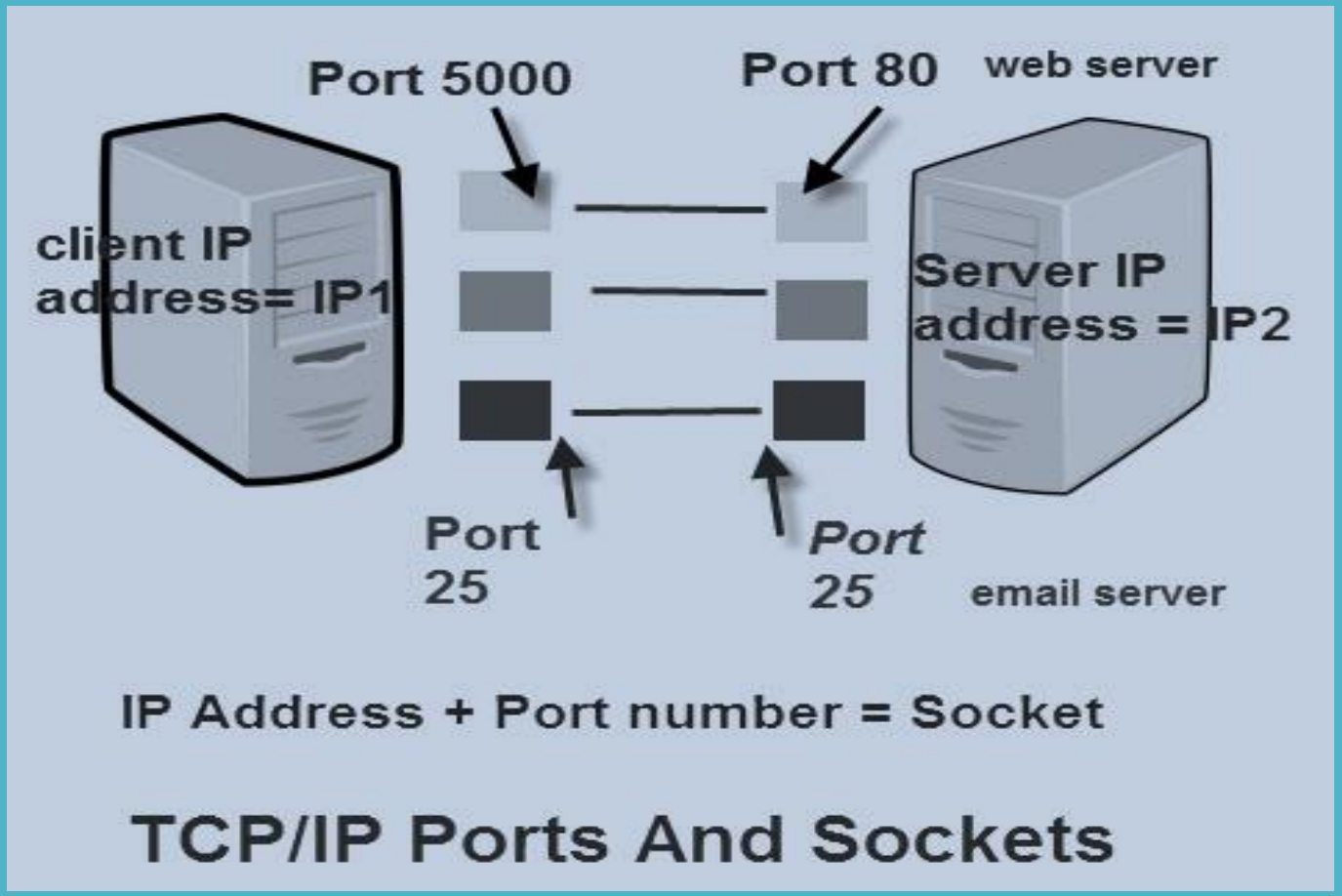
Υποδοχείς


Port 5000
Port 80
web server
client IP address= IP1
Server IP address = IP2
Port 25
Port 25
email server
IP Address + Port number = Socket
TCP/IP Ports And Sockets

Υποδοχείς

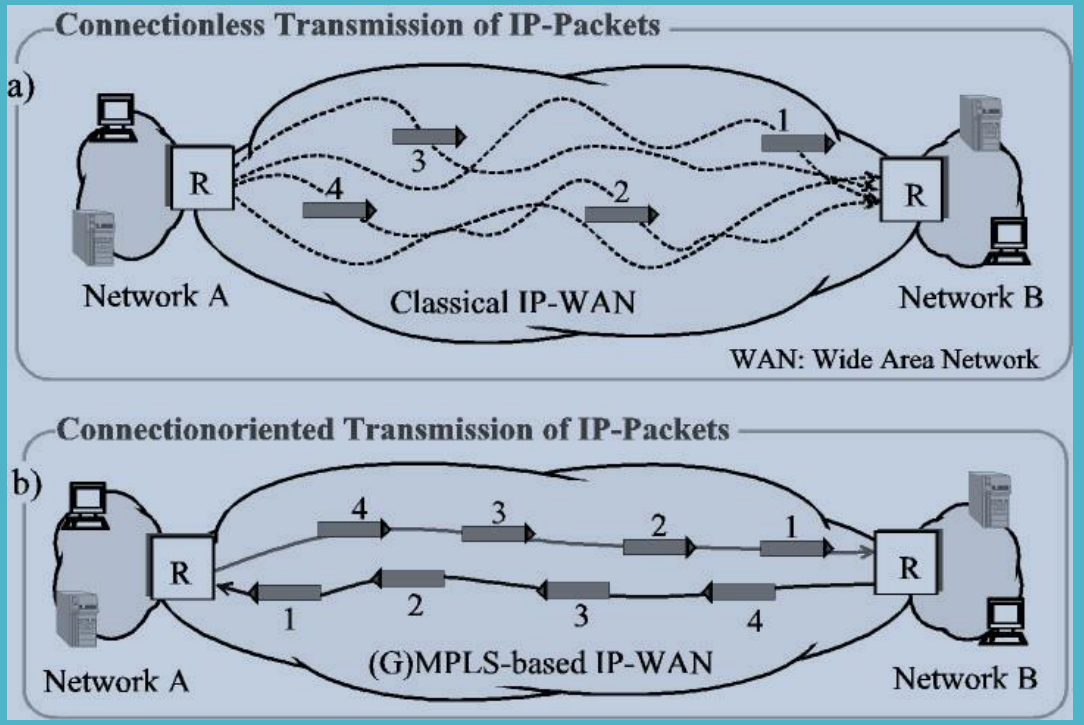
Οι υποδοχείς (sockets) είναι ειδικές δομές που επιτρέπουν την επικοινωνία ανάμεσα σε διεργασίες που εκτελούνται στον ίδιο ή σε διαφορετικούς υπολογιστές.

Η επικοινωνία μεταξύ υπολογιστών απαιτεί τη χρήση των κατάλληλων πρωτοκόλλων τα οποία είναι γνωστά ως δικτυακά πρωτόκολλα.

Η επικοινωνία μεταξύ υπολογιστών μπορεί να είναι τόσο με σύνδεση (connection oriented) όσο και χωρίς σύνδεση (connectionless).

Επικοινωνία με σύνδεση: η αποστολή των δεδομένων απαιτεί την αποκατάσταση της επικοινωνίας ανάμεσα στα δύο άκρα (τηλεφωνικό σύστημα).

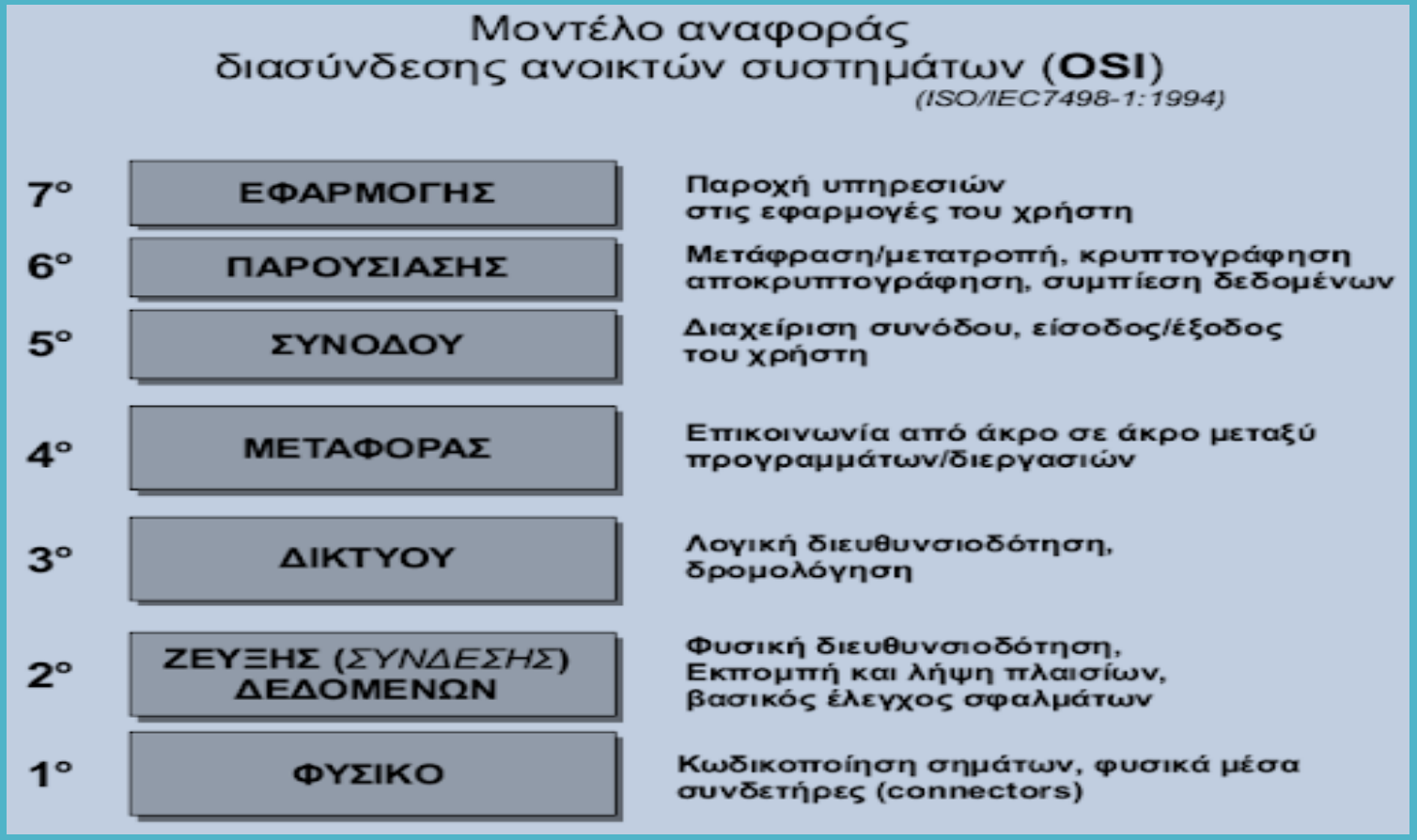
Επικοινωνία χωρίς σύνδεση: το κάθε πακέτο περιέχει τη διεύθυνση του παραλήπτη και δρομολογείται χωριστά και ανεξάρτητα από τα υπόλοιπα (ταχυδρομικό σύστημα.)



Υποδοχείς

Το μοντέλο αναφοράς OSI

Μοντέλο επτά επιπέδων – δεν χρησιμοποιείται ποτέ σε αυτή τη μορφή αλλά αποτελεί τη βάση και το πρότυπο πάνω στο οποίο στηρίζονται όλα τα δικτυακά μοντέλα αναφοράς

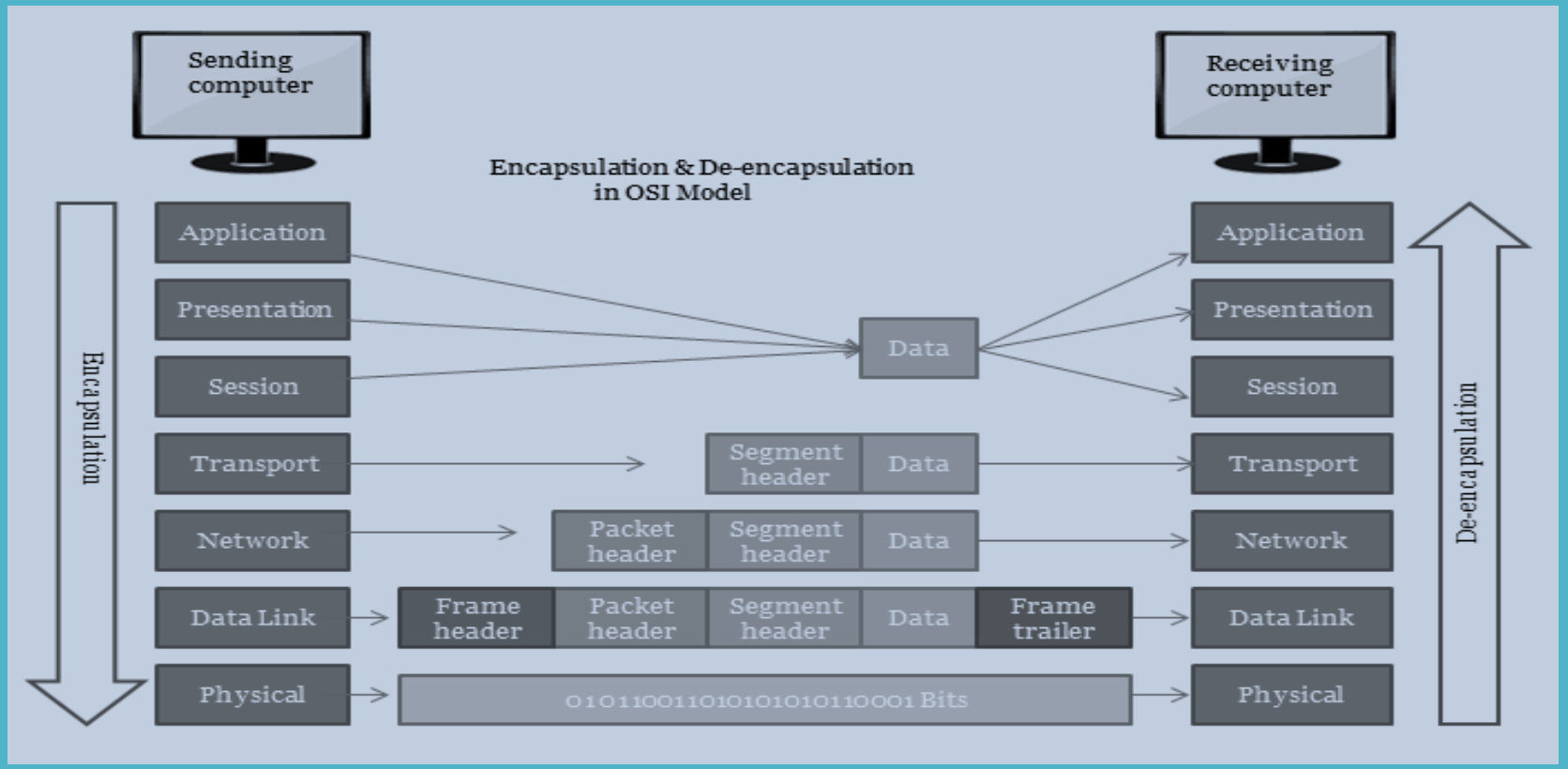


Υποδοχείς

Το μοντέλο αναφοράς OSI

Επικοινωνία υπολογιστών μέσω του μοντέλου αναφοράς OSI

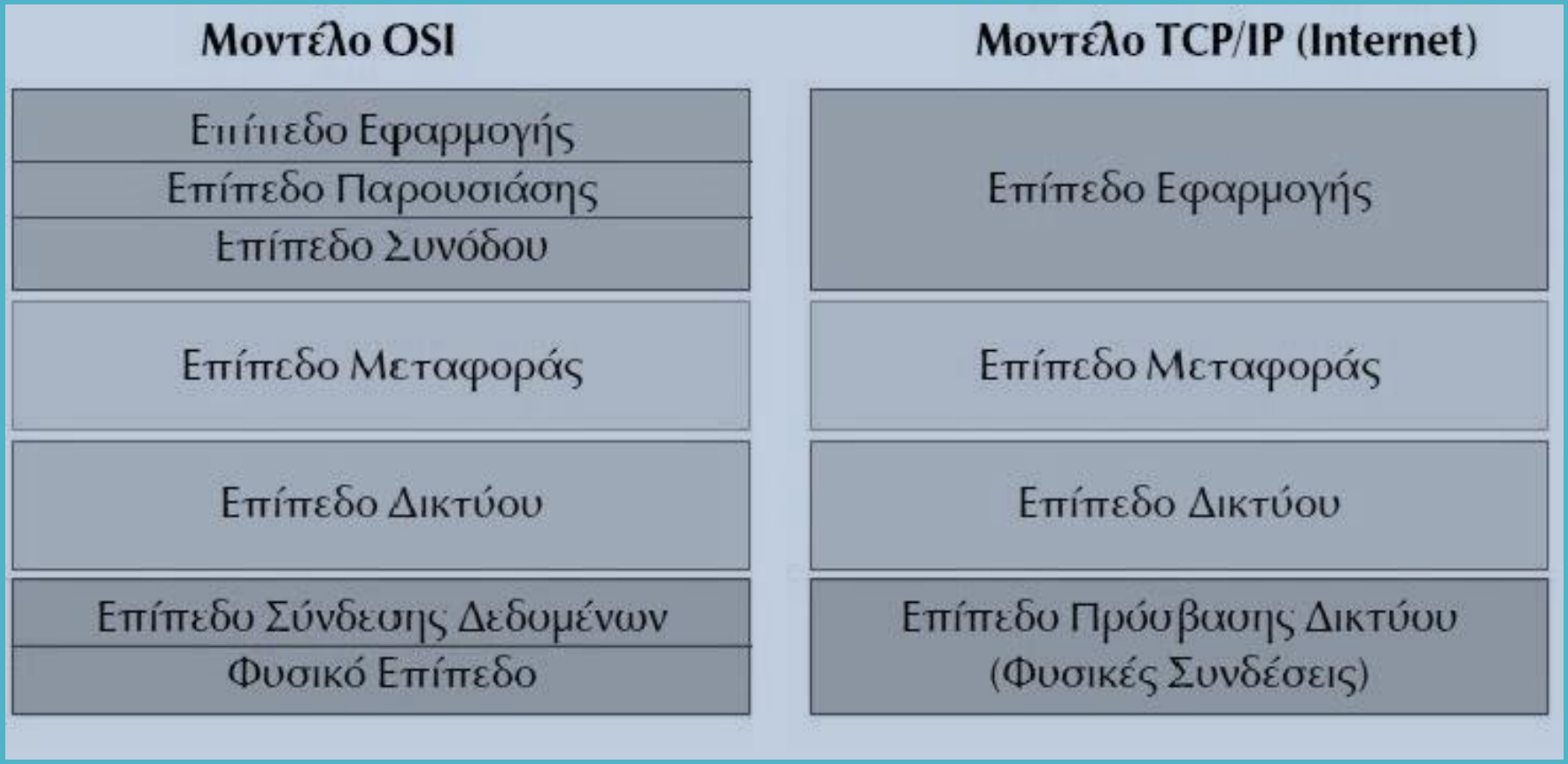
Στον υπολογιστή Α τα πακέτα κατεβαίνουν από το επίπεδο εφαρμογής προς το φυσικό επίπεδο, ενώ στον υπολογιστή Β κινούνται αντίστροφα. Κατά την κάθοδο, σε κάθε επίπεδο προστίθενται δεδομένα τα οποία αφαιρούνται κατά την άνοδο στον υπολογιστή – παραλήπτη (ενθυλάκωση δεδομένων).



Υποδοχείς

Το μοντέλο αναφοράς TCP / IP

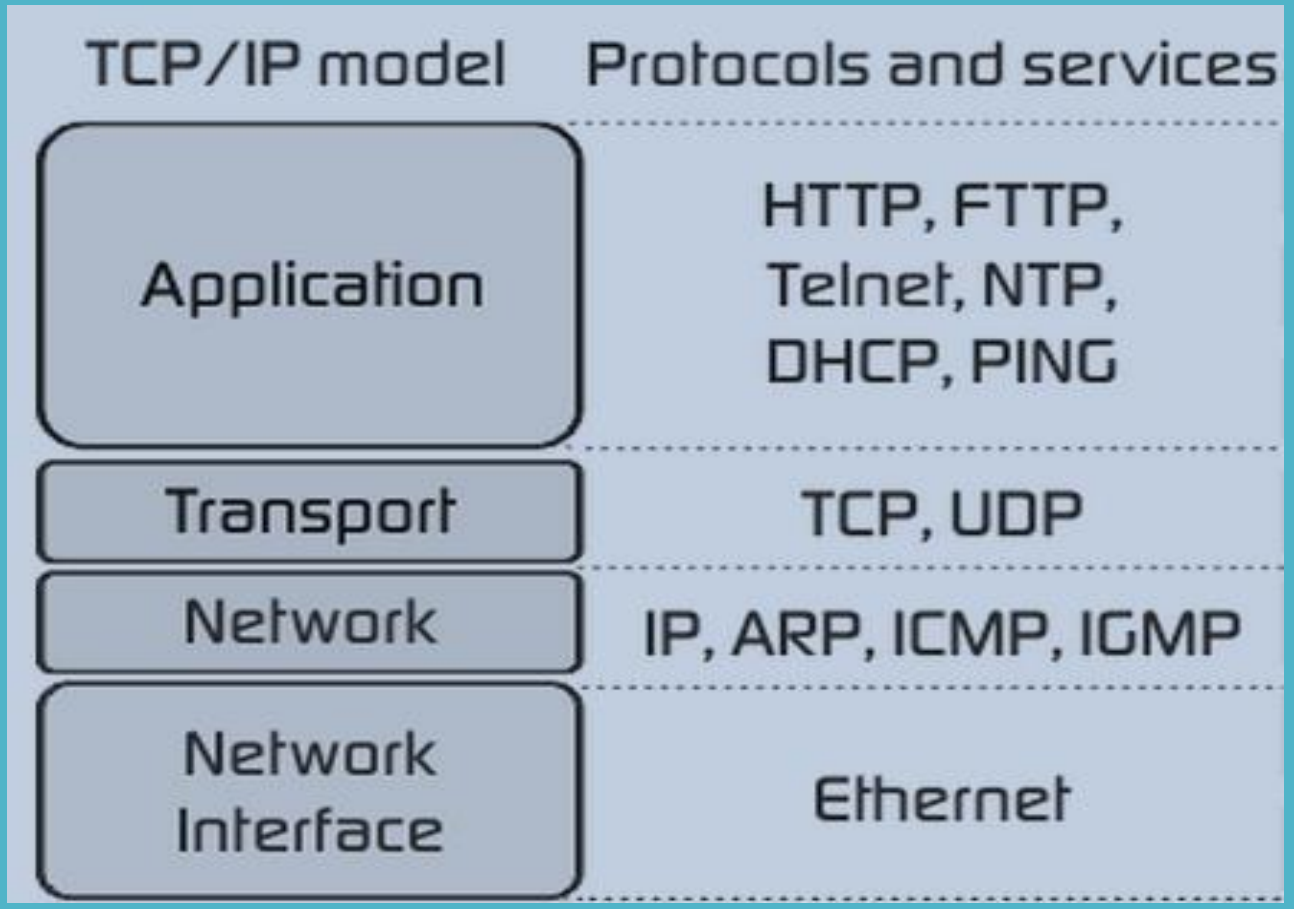
Αποτελεί το πιο διαδεδομένο δικτυακό πρωτόκολλο και συνιστά απαραίτητη προϋπόθεση για τη σύνδεση ενός υπολογιστή στο παγκόσμιο διαδίκτυο. Περιέχει τέσσερα επίπεδα.



Υποδοχείς

Το μοντέλο αναφοράς TCP / IP

Υπάρχουν πολλά πρωτόκολλα που μπορούν να ενταχθούν στα τέσσερα επίπεδα του TCP / IP και τα πιο σημαντικά από αυτά παρουσιάζονται στη συνέχεια.

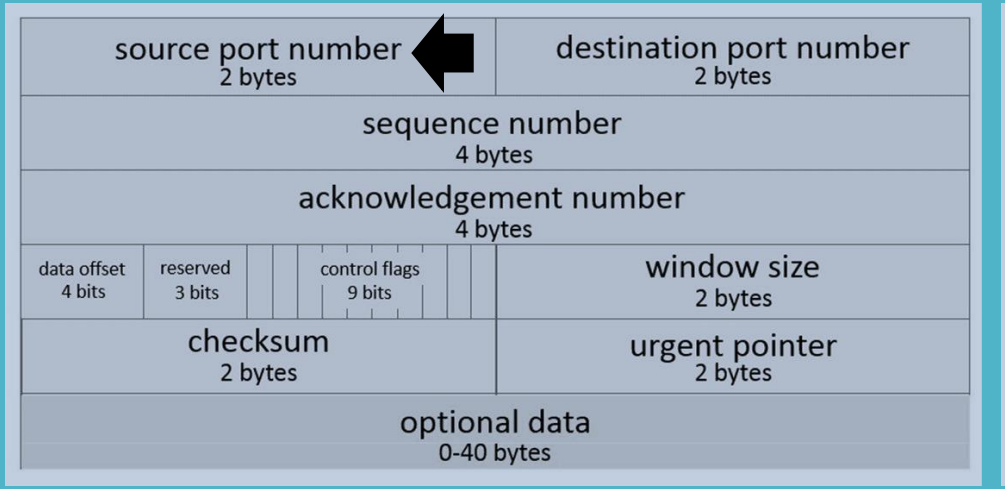


Υποδοχείς

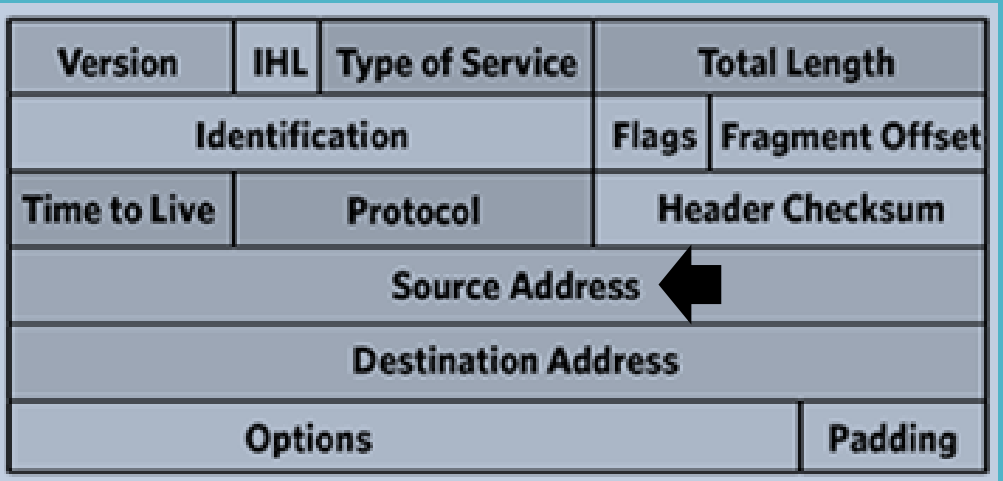
Το μοντέλο αναφοράς TCP / IP

Οι επικεφαλίδες που προστίθενται στα πακέτα δεδομένων στα επίπεδα μεταφοράς, δικτύου και πρόσβασης δικτύου (πρωτόκολλα TCP, IP και Ethernet).

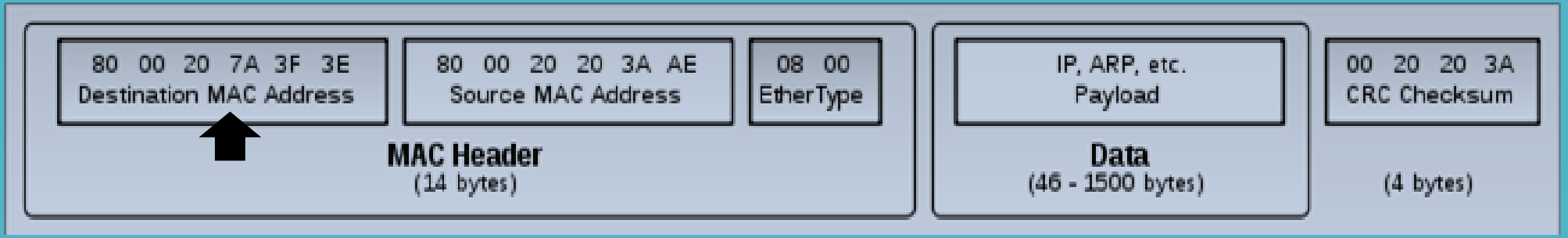
TCP Header



IP Header



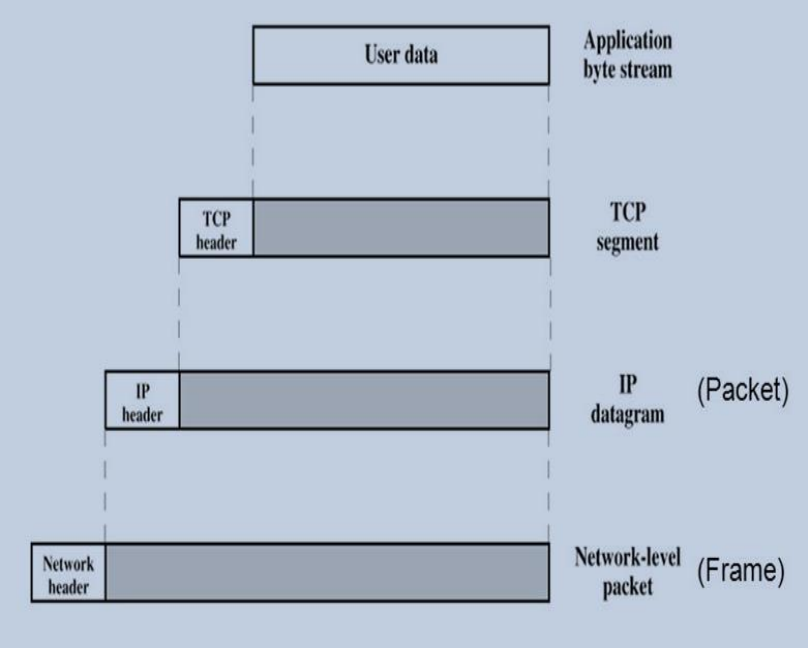
Ethernet header

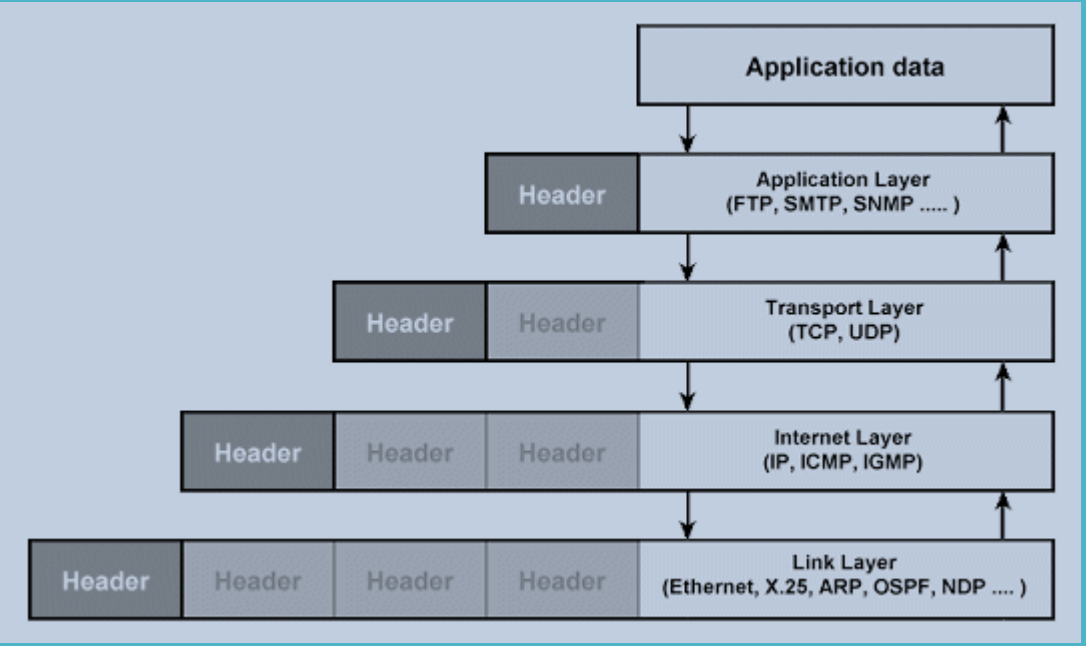


Υποδοχείς

Το μοντέλο αναφοράς TCP / IP

Το βασικό χαρακτηριστικό της ενθυλάκωσης δεδομένων είναι πως το κάθε επίπεδο δεν γνωρίζει την εσωτερική δομή που πακέτου που παραλαμβάνει – με άλλα λόγια το θεωρεί ως ένα ενιαίο πακέτο δεδομένων και απλά προσθέτει σε αυτό τη δική του επικεφαλίδα και τίποτε περισσότερο.

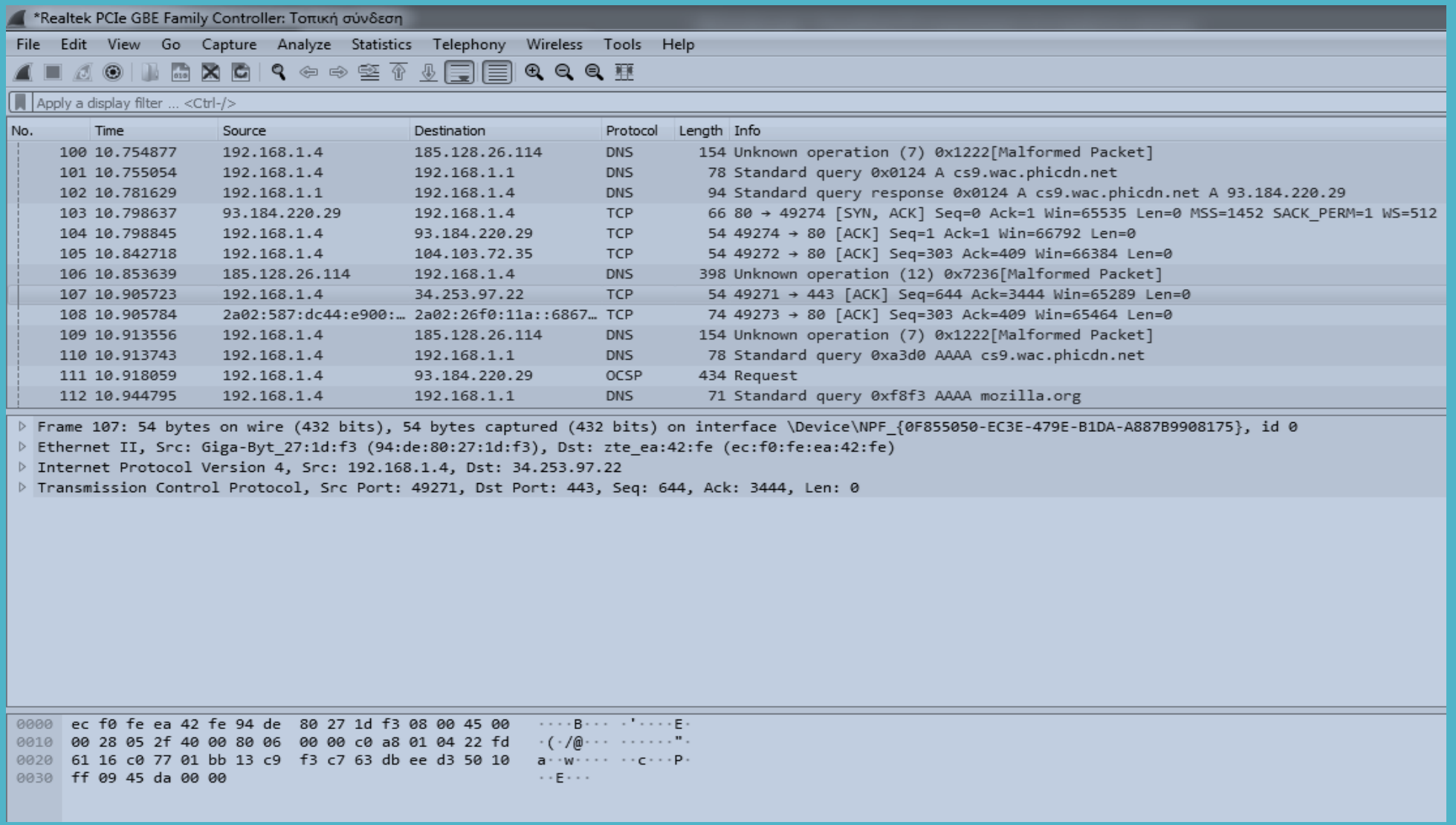




Για παράδειγμα, το IP δεν γνωρίζει πως μέσα στο TCP Segment υπάρχει TCP header - το μόνο που κάνει είναι να προσθέσει το δικό του header και να το προωθήσει στο επόμενο επίπεδο χωρίς να γνωρίζει (και στην πραγματικότητα, χωρίς να το ενδιαφέρει) τι υπάρχει μέσα στο TCP segment.

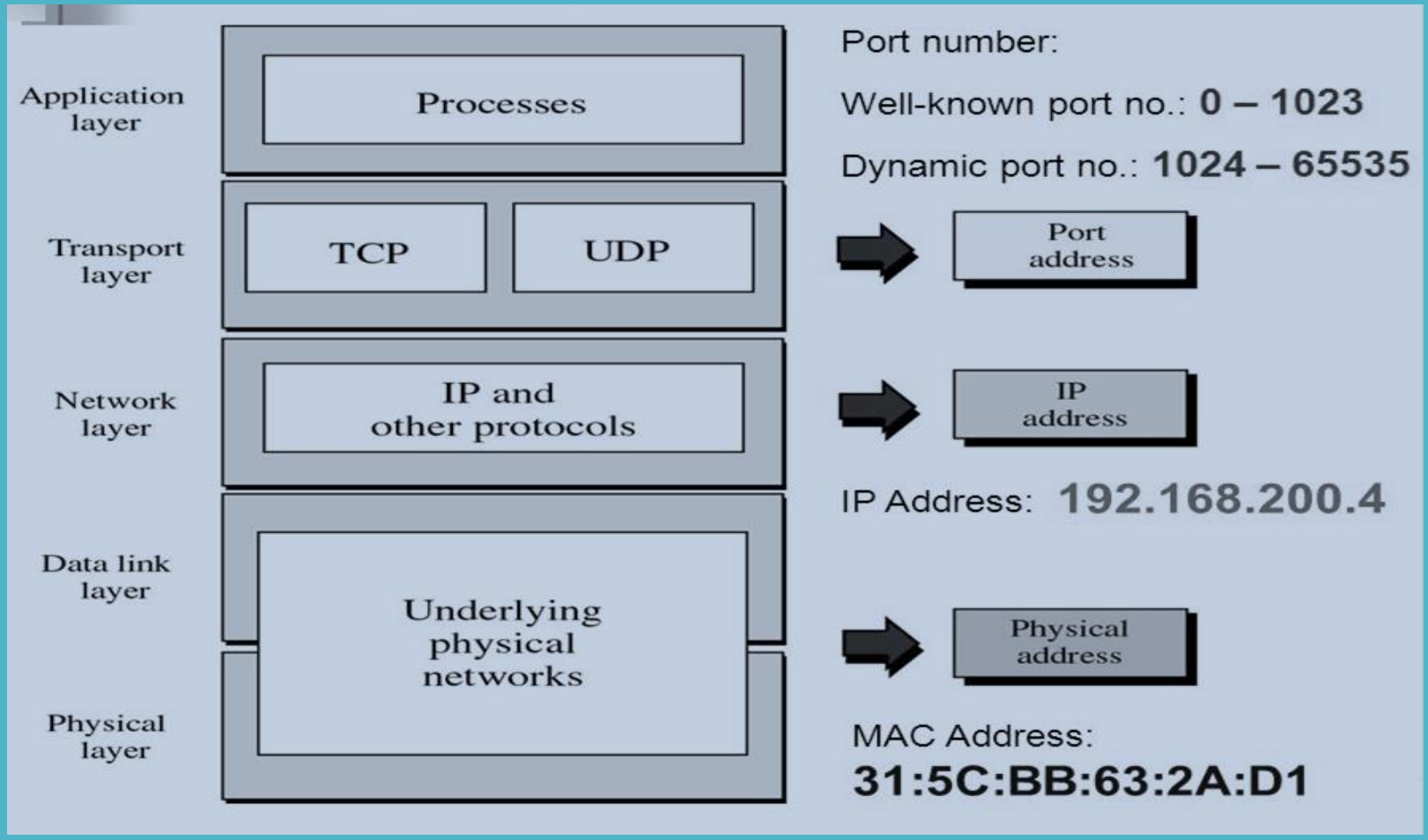
Υποδοχείς

Η βιβλιοθήκη npcap (παλαιότερα winpcap) και η εφαρμογή Wireshark



Υποδοχείς

Με πιο τρόπο είναι δυνατή η ταυτοποίηση ενός υπολογιστή σε κάθε επίπεδο? Αυτή η ταυτοποίηση απαιτείται στο επίπεδο μεταφοράς (port number), στο επίπεδο δικτύου (IP address) και στο επίπεδο πρόσβασης δικτύου (MAC address).



Υποδοχείς

Ανάκτηση των πληροφοριών διαμόρφωσης στο λειτουργικό σύστημα Linux – η εντολή ifconfig

```
amarg@amarg-vbox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::2079:b6d9:76c6:8708  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:59:2e:04  txqueuelen 1000  (Ethernet)
        RX packets 243  bytes 219625 (219.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 181  bytes 17403 (17.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 76  bytes 7294 (7.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 76  bytes 7294 (7.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

internet

loopback

Υποδοχείς

TCP / IP sockets vs Unix sockets

Ένα TCP / IP socket είναι ένας συνδυασμός της μορφής

IP Address : Port Number

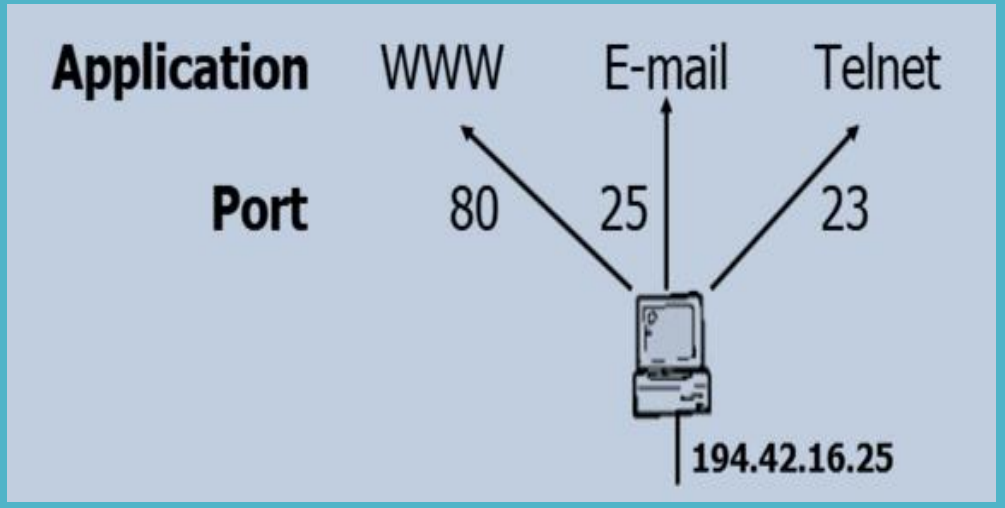
για παράδειγμα,

194.42.16.25 : 21

Στον παραπάνω συνδυασμό, η διεύθυνση IP προσδιορίζει με μοναδικό τρόπο έναν και μοναδικό υπολογιστή του παγκόσμιου διαδικτύου, ενώ ο αριθμός θύρας προσδιορίζει μία υπηρεσία σε αυτόν τον υπολογιστή.

Το σύστημα χρησιμοποιεί τα ports 0 – 1023 ενώ τα υπόλοιπα (με τιμές μέχρι 65536) χρησιμοποιούνται από τις εφαρμογές των χρηστών. Γνωστοί αριθμοί θύρας είναι

www → 80
ftp → 21
email → 25
ssh → 22
telnet → 23



Υποδοχείς

TCP / IP sockets vs Unix sockets

Ένα UNIX socket ή IPC (Interprocess communication) socket αποτελεί έναν μηχανισμό επικοινωνίας ανάμεσα σε διεργασίες που εκτελούνται στον ίδιο υπολογιστή που επιτρέπει αμφίδρομη επικοινωνία και ανταλλαγή δεδομένων.

Χρησιμοποιούνται με παρόμοιο τρόπο με τα TCP / IP sockets αλλά δεν στηρίζονται σε κάποιο υποκείμενο δικτυακό πρωτόκολλο που να επιτρέπει την επικοινωνία ανάμεσα σε δύο υπολογιστές – αντίθετα όλη η επικοινωνία πραγματοποιείται μέσα στον πυρήνα του λειτουργικού συστήματος.

Οι διεργασίες του συστήματος αντιλαμβάνονται τα UNIX sockets ως i-nodes και κατά συνέπεια, δύο διεργασίες μπορούν να επικοινωνήσουν μεταξύ τους , ανοίγοντας το ίδιο socket.

Τα UNIX sockets γνωρίζουν πως εκτελούνται στον ίδιο υπολογιστή και για το λόγο αυτό δεν καταφεύγουν σε πολύπλοκες διαδικασίες και ελέγχους όπως η δρομολόγηση και ο έλεγχος σφαλμάτων, γεγονός που τα καθιστά πιο αποτελεσματικά από τα TCP / IP sockets. Ωστόσο, υπόκεινται στον ίδιο έλεγχο πρόσβασης με τα συστήματα αρχείων, ενώ τα TCP / IP sockets υφίστανται έλεγχο σε επίπεδο φίλτρου πακέτου.

Εάν γνωρίζουμε πως η εφαρμογή μας θα εκτελεστεί σε έναν απλό υπολογιστή, είναι προτιμότερο να χρησιμοποιήσουμε UNIX sockets παρά TCP / IP sockets.

Υποδοχείς

TCP / IP sockets vs Unix sockets

Στα Unix sockets οι διευθύνσεις είναι ονόματα διαδρομής που δημιουργούνται στο σύστημα αρχείων όταν ένα socket συνδέεται στο όνομα διαδρομής. Αυτά τα αρχεία δεν ανοίγουν με τη συνάρτηση open αλλά με την κατάλληλη συνάρτηση της βιβλιοθήκης διαχείρισης των sockets.

Τα Unix sockets υποστηρίζουν τόσο STREAMS όσο και DGRAMS με τον πρώτο τύπο σύνδεσης να μοιάζει με τους αγωγούς (pipes) χωρίς όμως να ταυτίζεται μαζί τους.

Στην πραγματικότητα, τα Unix sockets πλεονεκτούν έναντι των αγωγών στο ότι είναι πλήρως αμφίδρομα και για το λόγο αυτό χρησιμοποιούνται για επικοινωνία μεταξύ διεργασιών (Interprocess Communication, IPC). Για τη διευκόλυνση του χρήστη προσφέρεται η συνάρτηση

int socketpair (int domain, int type, int protocol, int sockfds [2]);

με το τελευταίο όρισμά να περιέχει τους περιγραφείς αρχείων για τα δύο άκρα της σύνδεσης και τα υπόλοιπα ορίσματα να καθορίζουν τα χαρακτηριστικά του socket με τον τρόπο που θα δούμε σε λίγο.

Υποδοχείς

Δημιουργία socket

Η δημιουργία ενός socket γίνεται με την κλήση συστήματος socket η οποία δηλώνεται ως

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

όπου domain η οικογένεια πρωτοκόλλων επικοινωνίας που θα χρησιμοποιηθεί στην εντολή:

- **AF_UNIX** Local communication
- **AF_LOCAL** Synonym for **AF_UNIX**
- **AF_INET** IPv4 Internet protocols
- **AF_AX25** Amateur radio AX.25 protocol
- **AF_IPX** IPX - Novell protocols
- **AF_APPLETALK** AppleTalk
- **AF_X25** ITU-T X.25 / ISO-8208 protocol
- **AF_INET6** IPv6 Internet protocols
- **AF_DECnet** DECet protocol sockets
- **AF_KEY** Key management protocol
- **AF_NETLINK** Kernel user interface device
- **AF_PACKET** Low-level packet interface
- **AF_RDS** Reliable Datagram Sockets (RDS
- **AF_PPPOX** Generic PPP transport layer
- **AF_LLC** Logical link control (IEEE 802.2 LLC)
- **AF_IB** InfiniBand native addressing
- **AF_MPLS** Multiprotocol Label Switching
- **AF_CAN** Controller Area Network automotive bus protocol
- **AF_TIPC** TIPC, "cluster domain sockets" protocol
- **AF_BLUETOOTH** Bluetooth low-level socket protocol
- **AF_ALG** Interface to kernel crypto API
- **AF_VSOCK** VSOCK (originally "VMWare VSockets")
- **AF_KCM** KCM (kernel connection multiplexer) interface
- **AF_XDP** XDP (express data path) interface

Υποδοχείς

Δημιουργία socket

Το όρισμα type περιγράφει τον τύπο της επικοινωνίας και μπορεί να λάβει τις τιμές

SOCK_STREAM → προσφέρει αξιόπιστη επικοινωνία με σύνδεση με ρεύματα από bytes **(read / write ή send / recv).**
SOCK_DGRAM → προσφέρει αναξιόπιστη επικοινωνία χωρίς σύνδεση **(sendto / recvfrom)**.
SOCK_SEQPACKET → προσφέρει αξιόπιστη επικοινωνία με σύνδεση με ακολουθίας από αυτοδύναμα πακέτα.

Το όρισμα protocol περιγράφει το πρωτόκολλο της οικογένειας πρωτοκόλλων επικοινωνίας που θα χρησιμοποιηθεί. Η τιμή 0 ορίζει πως θα χρησιμοποιηθεί το προεπιλεγμένο πρωτόκολλο επικοινωνίας που είναι το TCP για επικοινωνία με σύνδεση και το UDP για επικοινωνία χωρίς σύνδεση.

TCP → Transmission Control Protocol
UDP → User Datagram Protocol

Τα ονόματα και οι κωδικοί των πρωτοκόλλων που μπορούν να χρησιμοποιηθούν, περιλαμβάνονται στο αρχείο διαμόρφωσης /etc/protocols.

Η εντολή socket επιστρέφει τιμή μικρότερη του μηδενός σε περίπτωση αποτυχίας, ενώ εάν η κλήση της πραγματοποιηθεί χωρίς πρόβλημα επιστρέφει έναν περιγραφέα αρχείου (file descriptor) με τιμή μεγαλύτερη ή ίση του μηδενός. Αυτή η τιμή ταυτοποιεί το socket σε όλες τις μεταγενέστερες διαδικασίες στις οποίες αυτό συμμετέχει.

Υποδοχείς

Τα περιεχόμενα του αρχείου /etc/protocols

```
ip      0       IP              # internet protocol, pseudo protocol number
hopopt  0       HOPOPT          # IPv6 Hop-by-Hop Option [RFC1883]
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # Internet Group Management
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
igp     9       IGP             # any private interior gateway (Cisco)
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
hmp     20      HMP             # host monitoring protocol
xns-idp 22      XNS-IDP         # Xerox NS IDP
rdp     27      RDP             # "reliable datagram" protocol
iso-tp4 29      ISO-TP4         # ISO Transport Protocol class 4 [RFC905]
dccp    33      DCCP            # Datagram Congestion Control Prot. [RFC4340]
xtp     36      XTP             # Xpress Transfer Protocol
ddp     37      DDP             # Datagram Delivery Protocol
idpr-cmtp 38    IDPR-CMTP       # IDPR Control Message Transport
ipv6    41      IPv6            # Internet Protocol, version 6
ipv6-route 43   IPv6-Route       # Routing Header for IPv6
ipv6-frag 44    IPv6-Frag       # Fragment Header for IPv6
idrp    45      IDRP            # Inter-Domain Routing Protocol
rsvp    46      RSVP            # Reservation Protocol
gre     47      GRE             # General Routing Encapsulation
esp     50      IPSEC-ESP       # Encap Security Payload [RFC2406]
ah      51      IPSEC-AH        # Authentication Header [RFC2402]
skip    57      SKIP            # SKIP
ipv6-icmp 58    IPv6-ICMP       # ICMP for IPv6
ipv6-nonxt 59   IPv6-NoNxt      # No Next Header for IPv6
ipv6-opts 60    IPv6-Opts       # Destination Options for IPv6
```

Υποδοχείς

Τα περιεχόμενα του αρχείου /etc/protocols

```
rspf    73      RSPF CPHB       # Radio Shortest Path First (officially CPHB)
vmtp    81      VMTP            # Versatile Message Transport
eigrp   88      EIGRP           # Enhanced Interior Routing Protocol (Cisco)
ospf    89      OSPFIGP         # Open Shortest Path First IGP
ax.25   93      AX.25           # AX.25 frames
ipip    94      IPIP            # IP-within-IP Encapsulation Protocol
etherip 97      ETHERIP         # Ethernet-within-IP Encapsulation [RFC3378]
encap   98      ENCAP           # Yet Another IP encapsulation [RFC1241]
#       99                      # any private encryption scheme
pim     103     PIM             # Protocol Independent Multicast
ipcomp  108     IPCOMP          # IP Payload Compression Protocol
vrrp    112     VRRP            # Virtual Router Redundancy Protocol [RFC5798]
l2tp    115     L2TP            # Layer Two Tunneling Protocol [RFC2661]
isis    124     ISIS            # IS-IS over IPv4
sctp    132     SCTP            # Stream Control Transmission Protocol
fc      133     FC              # Fibre Channel
mobility-header 135 Mobility-Header # Mobility Support for IPv6 [RFC3775]
udplite 136     UDPLite         # UDP-Lite [RFC3828]
mpls-in-ip 137  MPLS-in-IP      # MPLS-in-IP [RFC4023]
manet   138                     # MANET Protocols [RFC5498]
hip     139     HIP             # Host Identity Protocol
shim6   140     Shim6           # Shim6 Protocol [RFC5533]
wesp    141     WESP            # Wrapped Encapsulating Security Payload
rohc    142     ROHC            # Robust Header Compression
```

Υποδοχείς

Ορισμός και ανάκτηση επιλογών για sockets

Ο ορισμός και η ανάκτηση των τιμών για τις παραμέτρους λειτουργίας ενός socket, πραγματοποιείται με τις συναρτήσεις setsockopt και getsockopt που δηλώνονται στο αρχείο <sys/socket.h>.

int **setsockopt** (int *s*, int *level*, int *optname*, const void* *optval*, int *optlen*);

int **getsockopt** (int *s*, int *level*, int *optname*, void* *optval*, int* *optlen*);

s: the socket the system call applies to

level: layer which handle the option

optname: identifies the option we are setting/getting

optval: the value taken by the option

optlen: the size of data structure *optval*

Οι τιμές του level είναι οι:

SOL_SOCKET (default)
IPROTO_IP (Πρωτόκολλο IP)
IPROTO_IPV6 (Πρωτόκολλο IPv6)
IPROTO_ICMP (Πρωτόκολλο ICMP)
IPROTO_RAW (RAW data protocol)
IPROTO_TCP (Πρωτόκολλο TCP)
IPROTO_UDP (Πρωτόκολλο UDP)

Για το καθένα από τα παραπάνω πρωτόκολλα υπάρχει ξεχωριστή λίστα επιλογών και τιμών με τις λεπτομέρειες και τις σχετικές πληροφορίες να μπορούν να βρεθούν στα αρχεία τεκμηρίωσης.

Υποδοχείς

Ορισμός και ανάκτηση επιλογών για sockets

level	*optname*	get	set	Description	Flag	Datatype
SOL_SOCKET	SO_BROADCAST	•	•	Permit sending of broadcast datagrams	•	int
	SO_DEBUG	•	•	Enable debug tracing	•	int
	SO_DONTROUTE	•	•	Bypass routing table lookup	•	int
	SO_ERROR	•		Get pending error and clear		int
	SO_KEEPALIVE	•	•	Periodically test if connection still alive	•	int
	SO_LINGER	•	•	Linger on close if data to send		linger{}
	SO_OOBINLINE	•	•	Leave received out-of-band data inline	•	int
	SO_RCVBUF	•	•	Receive buffer size		int
	SO_SNDBUF	•	•	Send buffer size		int
	SO_RCVLOWAT	•	•	Receive buffer low-water mark		int
	SO_SNDLOWAT	•	•	Send buffer low-water mark		int
	SO_RCVTIMEO	•	•	Receive timeout		timeval{}
	SO_SNDTIMEO	•	•	Send timeout		timeval{}
	SO_REUSEADDR	•	•	Allow local address reuse	•	int
	SO_REUSEPORT	•	•	Allow local port reuse	•	int
	SO_TYPE	•		Get socket type		int
	SO_USELOOPBACK	•	•	Routing socket gets copy of what it sends	•	int
IPPROTO_IP	IP_HDRINCL	•	•	IP header included with data	•	int
	IP_OPTIONS	•	•	IP header options		(see text)
	IP_RECVDSTADDR	•	•	Return destination IP address	•	int
	IP_RECVIF	•	•	Return received interface index	•	int
	IP_TOS	•	•	Type-of-service and precedence		int
	IP_TTL	•	•	TTL		int
	IP_MULTICAST_IF	•	•	Specify outgoing interface		in_addr{}
	IP_MULTICAST_TTL	•	•	Specify outgoing TTL		u_char
	IP_MULTICAST_LOOP	•	•	Specify loopback		u_char
	IP_{ADD,DROP}_MEMBERSHIP		•	Join or leave multicast group		ip_mreq{}
	IP_{BLOCK,UNBLOCK}_SOURCE		•	Block or unblock multicast source		ip_mreq_source{}
	IP_{ADD,DROP}_SOURCE_MEMBERSHIP		•	Join or leave source-specific multicast		ip_mreq_source{}
IPPROTO_ICMPV6	ICMP6_FILTER	•	•	Specify ICMPv6 message types to pass		icmp6_filter{}

Υποδοχείς

Ορισμός και ανάκτηση επιλογών για sockets

IPPROTO_IPV6	IPV6_CHECKSUM	•	•	Offset of checksum field for raw sockets		int
	IPV6_DONTFRAG	•	•	Drop instead of fragment large packets	•	int
	IPV6_NEXTHOP	•	•	Specify next-hop address		sockaddr_in6()
	IPV6_PATHMTU	•		Retrieve current path MTU		ip6_mtuinfo()
	IPV6_RECVDSTOPTS	•	•	Receive destination options	•	int
	IPV6_RECVHOPLIMIT	•	•	Receive unicast hop limit	•	int
	IPV6_RECVHOPOPTS	•	•	Receive hop-by-hop options	•	int
	IPV6_RECVPATHMTU	•	•	Receive path MTU	•	int
	IPV6_RECVPKTINFO	•	•	Receive packet information	•	int
	IPV6_RECVRTHDR	•	•	Receive source route	•	int
	IPV6_RECVTCLASS	•	•	Receive traffic class	•	int
	IPV6_UNICAST_HOPS	•	•	Default unicast hop limit		int
	IPV6_USE_MIN_MTU	•	•	Use minimum MTU	•	int
	IPV6_V6ONLY	•	•	Disable v4 compatibility	•	int
	IPV6_XXX	•	•	Sticky ancillary data		(see text)
	IPV6_MULTICAST_IF	•	•	Specify outgoing interface		u_int
	IPV6_MULTICAST_HOPS	•	•	Specify outgoing hop limit		int
	IPV6_MULTICAST_LOOP	•	•	Specify loopback	•	u_int
	IPV6_JOIN_GROUP		•	Join multicast group		ipv6_mreq()
	IPV6_LEAVE_GROUP		•	Leave multicast group		ipv6_mreq()
IPPROTO_IP *or* IPPROTO_IPV6	MCAST_JOIN_GROUP		•	Join multicast group		group_req()
	MCAST_LEAVE_GROUP		•	Leave multicast group		group_source_req()
	MCAST_BLOCK_SOURCE		•	Block multicast source		group_source_req()
	MCAST_UNBLOCK_SOURCE		•	Unblock multicast source		group_source_req()
	MCAST_JOIN_SOURCE_GROUP		•	Join source-specific multicast		group_source_req()
	MCAST_LEAVE_SOURCE_GROUP		•	Leave source-specific multicast		group_source_req()

Υποδοχείς

Ορισμός και ανάκτηση επιλογών για sockets

level	*optname*	get	set	Description	Flag	Datatype
IPPROTO_TCP	TCP_MAXSEG	•	•	TCP maximum segment size		int
	TCP_NODELAY	•	•	Disable Nagle algorithm	•	int
IPPROTO_SCTP	SCTP_ADAPTION_LAYER	•	•	Adaption layer indication		sctp_setadaption{}
	SCTP_ASSOCINFO	†	•	Examine and set association info		sctp_assocparams{}
	SCTP_AUTOCLOSE	•	•	Autoclose operation		int
	SCTP_DEFAULT_SEND_PARAM	•	•	Default send parameters		sctp_sndrcvinfo{}
	SCTP_DISABLE_FRAGMENTS	•	•	SCTP fragmentation	•	int
	SCTP_EVENTS	•	•	Notification events of interest		sctp_event_subscribe{}
	SCTP_GET_PEER_ADDR_INFO	†		Retrieve peer address status		sctp_paddrinfo{}
	SCTP_I_WANT_MAPPED_V4_ADDR	•	•	Mapped v4 addresses	•	int
	SCTP_INITMSG	•	•	Default INIT parameters		sctp_initmsg{}
	SCTP_MAXBURST	•	•	Maximum burst size		int
	SCTP_MAXSEG	•	•	Maximum fragmentation size		int
	SCTP_NODELAY	•	•	Disable Nagle algorithm	•	int
	SCTP_PEER_ADDR_PARAMS	†	•	Peer address parameters		sctp_paddrparams{}
	SCTP_PRIMARY_ADDR	†	•	Primary destination address		sctp_setprim{}
	SCTP_RTOINFO	†	•	RTO information		sctp_rtoinfo{}
	SCTP_SET_PEER_PRIMARY_ADDR		•	Peer primary destination address		sctp_setpeerprim{}
	SCTP_STATUS	†		Get association status		sctp_status{}

Υποδοχείς

Αρχιτεκτονικές Client - Server

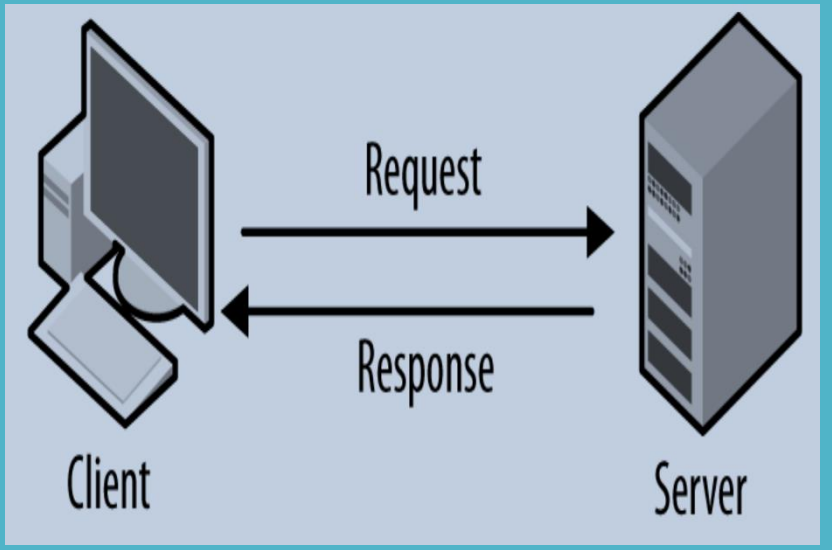
Στις αρχιτεκτονικές client – server υπάρχει ένας κεντρικός υπολογιστής (server) ο οποίος εξυπηρετεί αιτήσεις άλλων υπολογιστών (clients) που συνδέονται στον server μέσω δικτύου.

Ο client υποβάλλει στον server αιτήματα προς εξυπηρέτηση τα οποία διεκπεραιώνονται από τον server. Το αίτημα είτε εξυπηρετείται άμεσα είτε μπαίνει σε ουρά εκκρεμών μηνυμάτων εάν ο server είναι απασχολημένος.

Η αρχιτεκτονική client server έχει πολλά πλεονεκτήματα όπως:

1. Αποτελεσματική χρήση της υπολογιστικής ισχύος.
2. Μείωση του κόστους συντήρησης.
3. Αύξηση της παραγωγικότητας και της ευελιξίας.

Η επικοινωνία των client με τον server γίνεται μέσω socket.



ΠΑΡΑΔΕΙΓΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΩΝ CLIENT SERVER

1. Web Client – Web Server
2. FTP Client – FTP Server
3. Email Client – Email Server.

Υποδοχείς

Αρχιτεκτονικές Client - Server

Για την αποκατάσταση της σύνδεσης ο client και ο server λειτουργούν με διαφορετικό τρόπο:

Ειδικότερα:

Ο Client:

1. Δημιουργεί το socket καλώντας τη συνάρτηση socket.
2. Ενημερώνει το σύστημα σε ποια διεύθυνση θα συνδεθεί καλώντας τη συνάρτηση connect.

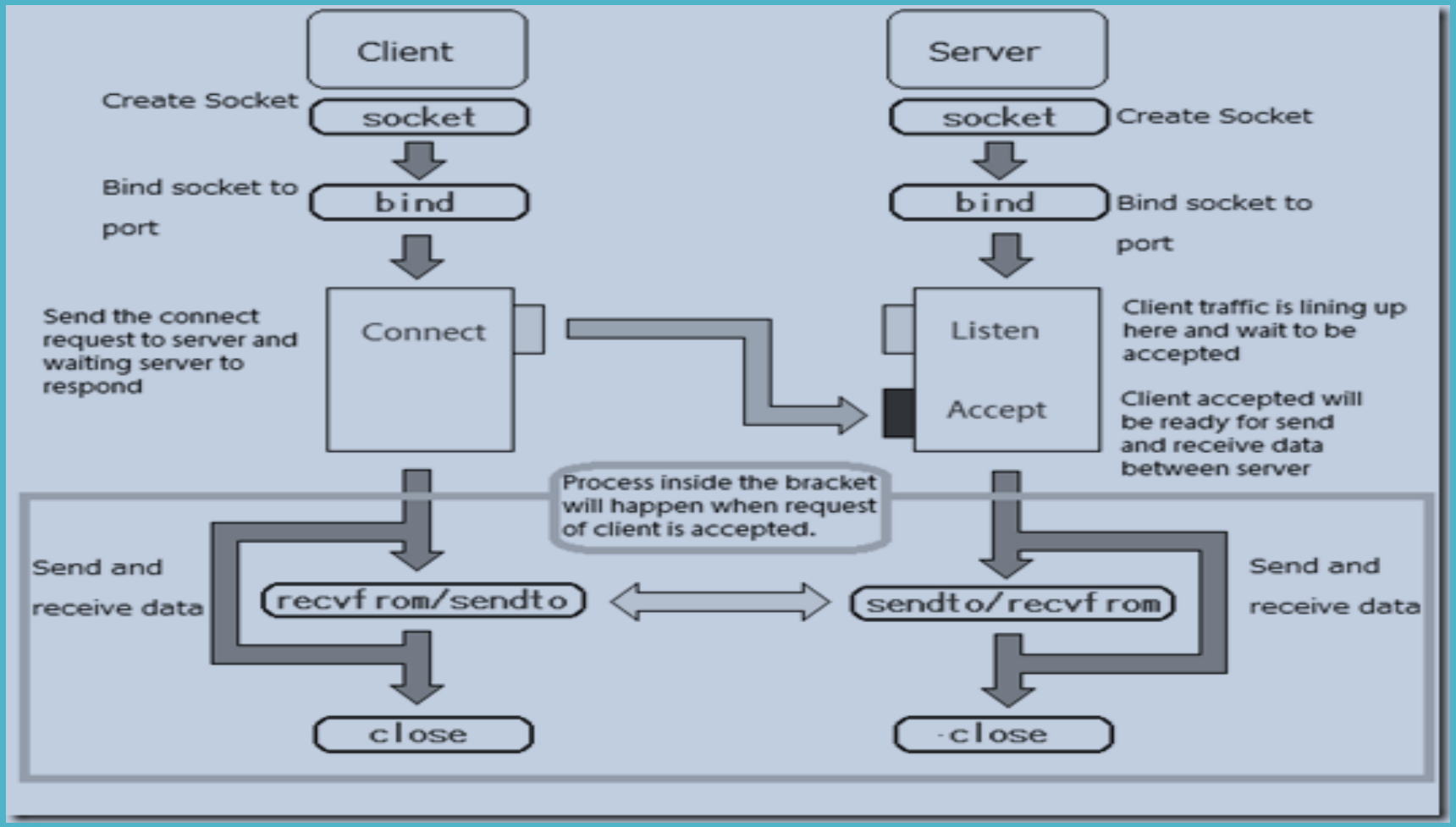
Ο Server:

1. Δημιουργεί το socket καλώντας τη συνάρτηση socket.
2. Συσχετίζει το socket που δημιούργησε με τη διεύθυνση στην οποία θα αναμένει αιτήματα σύνδεσης καλώντας τη συνάρτηση bind.
3. Αναμένει αιτήματα σύνδεσης καλώντας τη συνάρτηση listen.
4. Αποδέχεται αιτήματα σύνδεσης καλώντας τη συνάρτηση accept.

Υποδοχείς

Αρχιτεκτονικές Client - Server

Η αλληλεπίδραση του client με τον server με διαγραμματικό τρόπο απεικονίζεται στη συνέχεια.



Υποδοχείς

SOCKET PROGRAMMING

Η βασική δομή δεδομένων που χρησιμοποιείται με τις παραπάνω συναρτήσεις είναι η

```
struct sockaddr {
  uint8_t       sa_len;
  sa_family_t   sa_family;    /* address family: AF_xxx value */
  char          sa_data[14];  /* protocol-specific address */
};
```

(MAXSOCKADDRDATA=14) όπου sa_family η οικογένεια πρωτοκόλλων (π.χ. AF_UNIX, AF_INET) και sa_data δεδομένα που σχετίζονται με τη διεύθυνση του πρωτοκόλλου.

Η παραπάνω δομή είναι εντελώς γενική και χρησιμοποιείται με κάθε οικογένεια πρωτοκόλλων, ενώ για την ειδική περίπτωση των οικογενειών AF_UNIX και AF_INET μπορούν να χρησιμοποιηθούν οι δομές

UNIX SOCKET

```
struct sockaddr_un{
 uint8_t  sun_len;
 sa_family_t  sun_family;
 char  sun_path[104];   };
```

TCP/IP SOCKET

```
struct sockaddr_in {
   uint8_t          sin_len;
   sa_family_t      sin_family;
   in_port_t        sin_port;
   };  /* included in <netinet/in.h> */
```

όπου

```
struct in_addr { unsigned long s_addr; };
```

Υποδοχείς

SOCKET PROGRAMMING

Συσχέτιση του socket που δημιουργήθηκε, με την τοπική διεύθυνση για την επικοινωνία.

int bind (int sock, const struct sockaddr * addr, int addrlen);

Η bind συσχετίζει την τοπική διεύθυνση addr (η οποία θα πρέπει προηγουμένως να έχει αρχικοποιηθεί) με την ακέραια μεταβλητή sock που έχει επιστραφεί από την κλήση συστήματος socket.

Η bind καλείται υποχρεωτικά από την εφαρμογή του server προκειμένου αυτός να είναι σε θέση να δεχθεί αιτήματα σύνδεσης από την εφαρμογή του client ενώ αν και μπορεί να κληθεί και από την εφαρμογή του client αυτό δεν γίνεται σχεδόν ποτέ (γιατί δεν εξυπηρετεί σε τίποτε).

Εάν δεν συσχετίσουμε ένα socket μέσω της bind το λειτουργικό του εκχωρεί προσωρινά και για πολύ μικρό χρονικό διάστημα μία θύρα για την επίτευξη της επικοινωνίας.

Συνήθως δεν επιτρέπεται η συσχέτιση ενός socket με μία θύρα που ήδη χρησιμοποιείται.

Αν και οι δομές που χρησιμοποιούνται με AF_UNIX και AF_INET είναι αντίστοιχα οι sockaddr_un και sockaddr_in η bind (και η connect) απαιτούν δομή sockaddr – απαιτείται λοιπόν type casting.

Υποδοχείς

SOCKET PROGRAMMING

Αναμονή για αίτημα σύνδεσης πελάτη

int listen (int sock, int backlog);

Στην παραπάνω συνάρτηση που επίσης καλείται από τον server, το socket που περιγράφεται από το όρισμα sock που επέστρεψε η συνάρτηση socket, είναι ένα passive socket αφού το μόνο που κάνει είναι να αναμένει αιτήματα σύνδεσης πελατών προκειμένου να τα διεκπεραιώσει.

Τα αιτήματα σύνδεσης πελατών που δεν εξυπηρετούνται εν τη γενέσει τους χαρακτηρίζονται ως εκκρεμή και τοποθετούνται στο τέλος μιας ουράς αναμονής εξυπηρέτησης.

Αυτή η ουρά δεν είναι δυνατόν να αυξάνει απεριόριστα αλλά έχει πεπερασμένο μήκος το οποίο προσδιορίζεται από το δεύτερο όρισμα backlog η τυπική τιμή του οποίου είναι ίση με 5.

Εάν η ουρά αναμονής είναι πλήρης, ο πελάτης λαμβάνει το μήνυμα σφάλματος ECONNREFUSED ενώ σε περίπτωση επιτυχούς συσχέτισης η συνάρτηση επιστρέφει μηδενική τιμή.

Υποδοχείς

SOCKET PROGRAMMING

Υποβολή αιτήματος σύνδεσης από πελάτη

int connect (int sock, const struct sockaddr * addr, int addrlen);

Αυτή η συνάρτηση χρησιμοποιείται για την υποβολή ενός αιτήματος σύνδεσης του πελάτη και ένα άλλο TCP socket το οποίο βρίσκεται στην πλευρά του server.

Στα connection based sockets (SOCK_STREAM) αυτή η σύνδεση πραγματοποιείται μία και μοναδική φορά, ενώ στα connectionless sockets (SOCK_DGRAM) η συνάρτηση connect μπορεί να κληθεί πολλές φορές για να συνδεθεί κάθε φορά με διαφορετικό socket.

Όταν ο πελάτης υποβάλλει ένα αίτημα σύνδεσης, αυτό τοποθετείται στην ουρά αναμονής αιτημάτων εξυπηρέτησης που διατηρείται στην πλευρά του server όπου και παραμένει μέχρι ο server να κάνει αποδεκτό το αίτημα σύνδεσης.

Η connect αναστέλλει τη λειτουργία της μέχρι να γίνει αποδεκτό το αίτημα της σύνδεσης που υπέβαλλε. Η συνάρτηση επιστρέφει μηδέν σε περίπτωση επιτυχία και -1 σε περίπτωση αποτυχίας, αρχικοποιώντας με τον κατάλληλο τρόπο και τον κωδικό σφάλματος.

Υποδοχείς

SOCKET PROGRAMMING

Εξυπηρέτηση αιτήματος σύνδεσης πελάτη

int accept (int sock, struct sockaddr * addr, int addrlen);

Αυτή η συνάρτηση χρησιμοποιείται με connection based sockets (SOCK_STREAM) και δημιουργεί μία σύνδεση εξυπηρέτησης για το εκκρεμές αίτημα σύνδεσης που βρίσκεται στην κορυφή της ουράς αναμονής και σχετίζεται με το ενεργό socket που περιγράφεται από το πρώτο όρισμα.

Η accept αναστέλλει τη λειτουργία της μέχρι να φτάσει αίτημα σύνδεσης με κλήση της connect από κάποιο πελάτη.

Όταν λάβει αυτό ένα τέτοιο αίτημα:

Δημιουργεί ένα νέο active socket που συνομιλεί με το active socket του πελάτη (που είναι διαφορετικό από το αρχικό passive socket που χρησιμοποιήθηκε ως όρισμα στη listen).

Επιστρέφει τον file descriptor που αναφέρεται στο νέο αυτό socket.

Υποδοχείς

SOCKET PROGRAMMING

Ανταλλαγή μηνυμάτων στα Unix sockets

Στην περίπτωση κατά την οποία χρησιμοποιούνται για την επικοινωνία Unix sockets, η αποστολή και η ανταλλαγή δεδομένων πραγματοποιούνται με τις συναρτήσεις read και write αλλά και τις

int sendmsg (int fd, struct msghdr * msg, unsigned int flags);

int recvmsg (int fd, struct msghdr * msg, unsigned int flags);

όπου fd ο περιγραφέας αρχείου μέσω του οποίου αποστέλλεται ή παραλαμβάνεται η πληροφορία, msghdr ειδική δομή δεδομένων που περιγράφει το μήνυμα, ενώ η τιμή του flags είναι συνήθως 0.

```
struct msghdr
        {
        void            *msg_name;      // optional address
        socklen_t       msg_namelen;    // size of address
        struct iovec    *msg_iov;       // scatter/gather array
        int             msg_iovlen;     // no. of members
        void            *msg_control;   // ancillary data buffer
        int             msg_controllen; // ancillary buffer length
        int             flags;          // flags on received message
        };
```

Υποδοχείς

SOCKET PROGRAMMING

Παράδειγμα χρήσης Unix socket A

Ο server λειτουργεί επαναληπτικά και επιτρέπει τη σύνδεση ενός πελάτη κάθε φορά. Ο πελάτης συνδέεται στο socket του server και αντιγράψει απλά την είσοδό του στον server.

SERVER

```c
int main(void) {
    struct sockaddr_un address;
    int sock, conn;
    socklen_t addrLength;
    if ((sock = socket(PF_UNIX, SOCK_STREAM, 0)) < 0) {
         perror("Message from socket:"); exit(1); }
    unlink("./sample-socket");
    address.sun_family = AF_UNIX;
    strcpy(address.sun_path, "./sample-socket");
    addrLength = sizeof(address.sun_family) + strlen(address.sun_path);

    if (bind(sock, (struct sockaddr *) &address, addrLength)){
        perror("Message from bind:"); exit(1); }
    if (listen(sock, 5)){
        perror("Message from listen:"); exit(1); }
    while ((conn = accept(sock, (struct sockaddr *) &address,
                          &addrLength)) >= 0) {
        printf("---- getting data\n");
        copyData(conn, 1);
        printf("---- done\n");
        close(conn); }
    if (conn < 0) {
        perror("Message from accept:"); exit(1); }
    close(sock);
    return 0; }
```

```c
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>
```

Η συνάρτηση copyData αντιγράφει δεδομένα από τον περιγραφέα from στον περιγραφέα to μέχρι να μην υπάρχει τίποτε άλλο να αντιγραφεί.

```c
void copyData(int from, int to) {
    char buf[1024];
    int amount;
    while ((amount = read(from, buf, sizeof(buf))) > 0) {
       if (write(to, buf, amount) != amount) {
            perror("Message from write:"); exit(1); }}
    if (amount < 0) {
        perror("Message from read:"); exit(1); } }
```

Υποδοχείς

SOCKET PROGRAMMING

Παράδειγμα χρήσης Unix socket A

CLIENT

```c
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>

void copyData(int from, int to) {
    char buf[1024];
    int amount;
    while ((amount = read(from, buf, sizeof(buf))) > 0) {
       if (write(to, buf, amount) != amount) {
            perror("Message from write:"); exit(1); }}
    if (amount < 0) {
        perror("Message from read:"); exit(1); } }

int main(void) {
    struct sockaddr_un address;
    int sock;
    socklen_t addrLength;
    if ((sock = socket(PF_UNIX, SOCK_STREAM, 0)) < 0){
         perror("Message from socket:"); exit(1); }
    address.sun_family = AF_UNIX;     /* Unix domain socket */
    strcpy(address.sun_path, "./sample-socket");
    addrLength = sizeof(address.sun_family) + strlen(address.sun_path);
    if (connect(sock, (struct sockaddr *) &address, addrLength)){
         perror("Message from connect:"); exit(1); }
    copyData(0, sock);
    close(sock);
    return 0; }
```

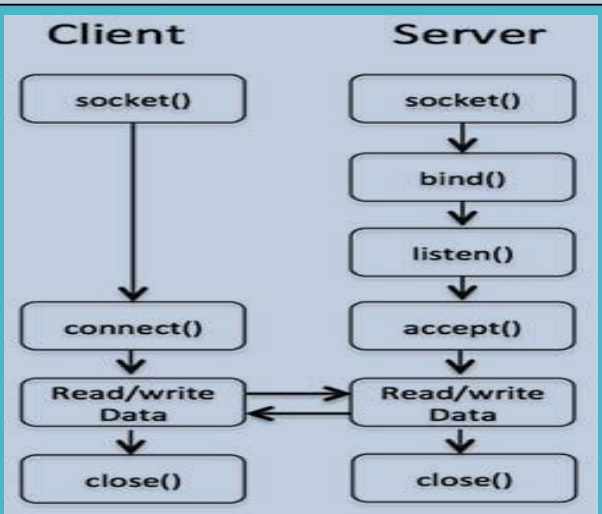
Υποδοχείς

SOCKET PROGRAMMING

Παράδειγμα χρήσης Unix socket A

CLIENT

```
int main(void) {
    struct sockaddr_un address;
    int sock;
    socklen_t addrLength;
    if ((sock = socket(PF_UNIX, SOCK_
         perror("Message from socket:"
    address.sun_family = AF_UNIX;
    strcpy(address.sun_path, "./sample
    addrLength = sizeof(address.sun_fa
    if (connect(sock, (struct sockaddr
         perror("Message from connect:
    copyData(0, sock);
    close(sock);
    return 0; }
```



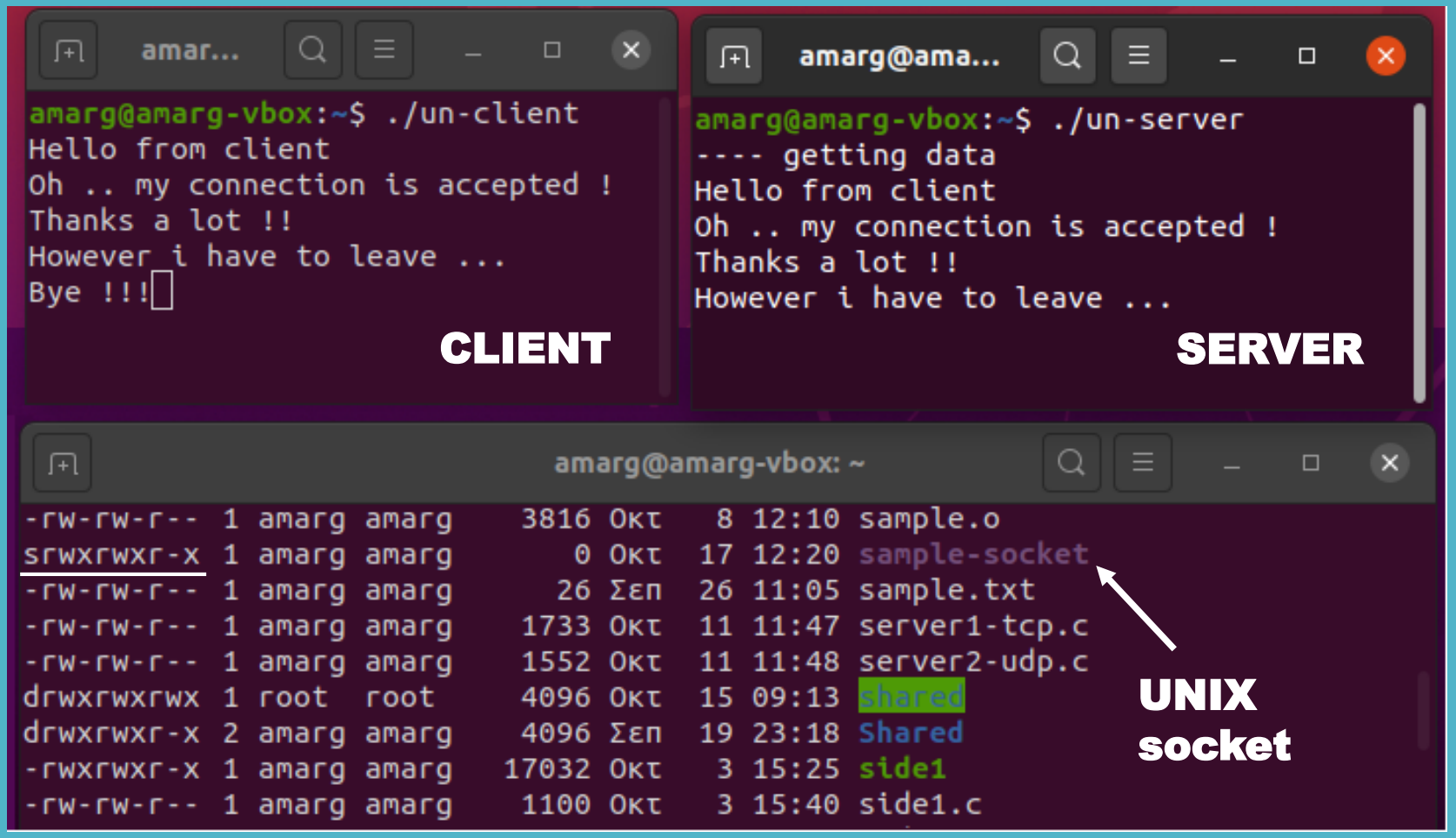
SERVER

```
int main(void) {
    struct sockaddr_un address;
    int sock, conn;
    socklen_t addrLength;
    if ((sock = socket(PF_UNIX, SOCK_STREAM, 0)) < 0) {
         perror("Message from socket:"); exit(1); }
    unlink("./sample-socket");
    address.sun_family = AF_UNIX;
    strcpy(address.sun_path, "./sample-socket");
    addrLength = sizeof(address.sun_family) + strlen(address.sun_path);

    if (bind(sock, (struct sockaddr *) &address, addrLength)){
        perror("Message from bind:"); exit(1); }
    if (listen(sock, 5)){
        perror("Message from listen:"); exit(1); }
    while ((conn = accept(sock, (struct sockaddr *) &address,
                          &addrLength)) >= 0) {
        printf("---- getting data\n");
        copyData(conn, 1);
        printf("---- done\n");
        close(conn); }
    if (conn < 0) {
        perror("Message from accept:"); exit(1); }
    close(sock);
    return 0; }
```

Υποδοχείς

SOCKET PROGRAMMING



Το UNIX socket αποτελεί ένα αρχείο του σκληρού δίσκου !!

Υποδοχείς

SOCKET PROGRAMMING

Παράδειγμα χρήσης Unix socket B

Ο server και ο client συνδέονται με μία σχέση πατέρα (server) – παιδιού (client) που δημιουργείται μέσω της fork και επικοινωνούν στέλνοντας ένα μήνυμα ο ένας στον άλλο.

```c
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>

#define SOCKETNAME "MySocket"

int main(void) {
    struct sockaddr_un sa;
    (void)unlink(SOCKETNAME);
    strcpy(sa.sun_path, SOCKETNAME);
    sa.sun_family = AF_UNIX;
    int fd_skt, fd_client; char buf[100];
    int written; ssize_t readb;
```

CLIENT

```c
if (fork() == 0) {
    if ((fd_skt = socket(PF_UNIX, SOCK_STREAM, 0)) < 0) {
         perror("Message from socket [client] : ");
         exit(-1); }
    printf ("[Client] ==> Socket %d has been created\n", fd_skt);
    while (connect(fd_skt, (struct sockaddr *)&sa, sizeof(sa)) == -1) {
        if (errno == ENOENT) {
            sleep(1); continue; }
        else {
         perror("Message from connect [client]");
         exit(-2); }}
    printf ("[Client] ==> Connection has been established .. let us write to server\n");
    written = write(fd_skt, "Hello!", 7);
    if (written==-1) {
         perror("Message from write [client]");
         exit(-3); }
    else printf ("[Client] ==> %d bytes written to server\n", written);
    printf ("[Client] ==> Now let us read from server\n");
    readb = read(fd_skt, buf, sizeof(buf));
    if (readb==-1) {
         perror("Message from read [client]");
         exit(-4); }
    else printf ("[Client] ==> %zd bytes read from server\n", readb);
    printf("Client got %s\n", buf);
    close(fd_skt);
    exit(0); }
```

Υποδοχείς

SOCKET PROGRAMMING

SERVER

```
else {
    if ((fd_skt = socket(PF_UNIX, SOCK_STREAM, 0)) < 0) {
         perror("Message from socket [server]");
    printf ("[Server] ==> Socket %d has been created\n", fd_skt);
    if (bind(fd_skt, (struct sockaddr *) &sa, sizeof(sa))){
        perror("Message from bind [server]");
        exit(-2); }
    printf ("[Server] ==> Socket %d has been bound to address\n", fd_skt);
    if (listen(fd_skt, 5)){
        perror("Message from listen [server]");
        exit(-3); }
    printf ("[Server] ==> Listening for incoming messages\n");
    if ((fd_client = accept(fd_skt, NULL, 0))<0) {
        perror("Message from accept [server]");
        exit(-4); }
    printf ("[Server] ==> let us read from client via socket %d\n", fd_client);
    readb = read(fd_client, buf, sizeof(buf));
    if (readb == -1) {
         perror("Message from read [server] : ");
         exit(-5); }
    printf("Server got %s ==> %zd bytes read from client\n", buf, readb);
    printf ("[Server] ==> let us write to client\n");
    if (write(fd_client, "Goodbye!", 9)==-1) {
         perror("Message from write [server]");
         exit(-6); }
    close(fd_skt);
    close(fd_client);
    exit(0); }}
```

Ο server και ο client συνδέονται με μία σχέση πατέρα (server) – παιδιού (client) που δημιουργείται μέσω της fork και επικοινωνούν στέλνοντας ένα μήνυμα ο ένας στον άλλο.

```
amarg@amarg-vbox:~$ ./fork-cl-srv
[Server] ==> Socket 3 has been created
[Server] ==> Socket 3 has been bound to address
[Server] ==> Listening for incoming messages
[Client] ==> Socket 3 has been created
[Client] ==> Connection has been established .. let us write to server
[Server] ==> let us read from client via socket 4
[Client] ==> 7 bytes written to server
Server got Hello! ==> 7 bytes read from client
[Server] ==> let us write to client
amarg@amarg-vbox:~$ [Client] ==> Now let us read from server
[Client] ==> 9 bytes read from server
Client got Goodbye!
```

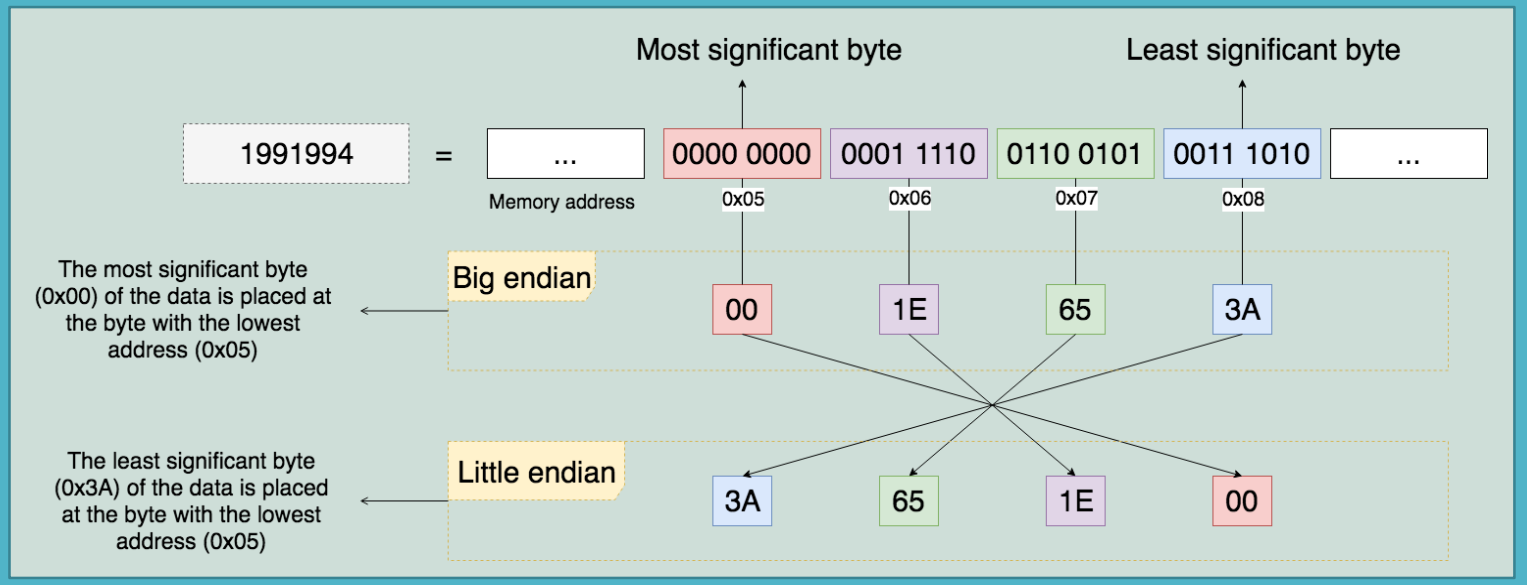
Υποδοχείς

SOCKET PROGRAMMING

Χρησιμοποιώντας TCP / IP sockets

Το βασικό χαρακτηριστικό των δικτύων που χρησιμοποιούν το πρωτόκολλο TCP / IP είναι πως αποτελούν ετερογενή δίκτυα, υπό την έννοια πως οι υπολογιστές που επικοινωνούν μέσα από αυτά μπορεί να διαφέρουν τόσο ως προς την αρχιτεκτονική όσο και ως προς το λειτουργικό σύστημα.

Μία βασική διαφορά εντοπίζεται στον τρόπο αποθήκευσης των bytes ενός αριθμού των 32 bits (4 bytes). Οι δύο βασικοί τρόποι αποθήκευσης είναι ο big endian (80x86) και ο little endian (SUN SPARC, Motorola, Power PC) που αποθηκεύουν τα bytes με την αντίστροφη σειρά και δεν είναι συμβατοί μεταξύ τους.



Υποδοχείς

SOCKET PROGRAMMING

Χρησιμοποιώντας TCP / IP sockets

Σε περιπτώσεις κατά τις οποίες οι υπολογιστές που επικοινωνούν χρησιμοποιούν διαφορετικό τρόπο αναπαράστασης (δηλαδή ο ένας χρησιμοποιεί big-endian [κωδικοποίηση πού είναι γνωστή και ως network byte order] και ο άλλος little-endian) τα δικτυακά πρωτόκολλα είναι υπεύθυνα για την πραγματοποίηση των κατάλληλων μετασχηματισμών δεδομένων έτσι ώστε οι δύο υπολογιστές να είναι σε θέση να επικοινωνήσουν μεταξύ τους.

Η μετατροπή από τον έναν τύπο κωδικοποίησης στον άλλο γίνεται από τις συναρτήσεις της γλώσσας C με ονόματα htonl, htons, ntohl και ntohs (netinet/in.h) και με ορίσματα αριθμούς μήκους 32 bits

```
unsigned int htonl (unsigned int hostlong);
unsigned short htons (unsigned short hostshort);
unsigned int ntohl (unsigned int netlong);
unsigned short ntohs (unsigned short netshort);
```

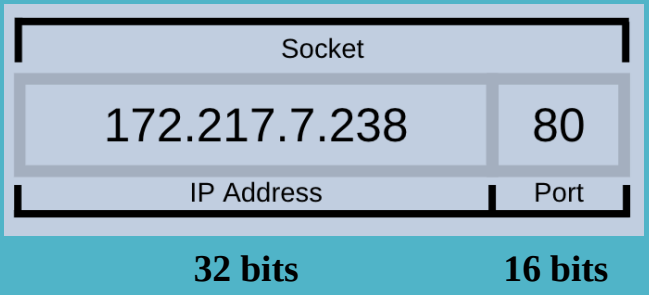
htonl → host to network, long
htons → host to network, short

ntohl → network to host, long
ntohs → network to host, short

Υποδοχείς

SOCKET PROGRAMMING

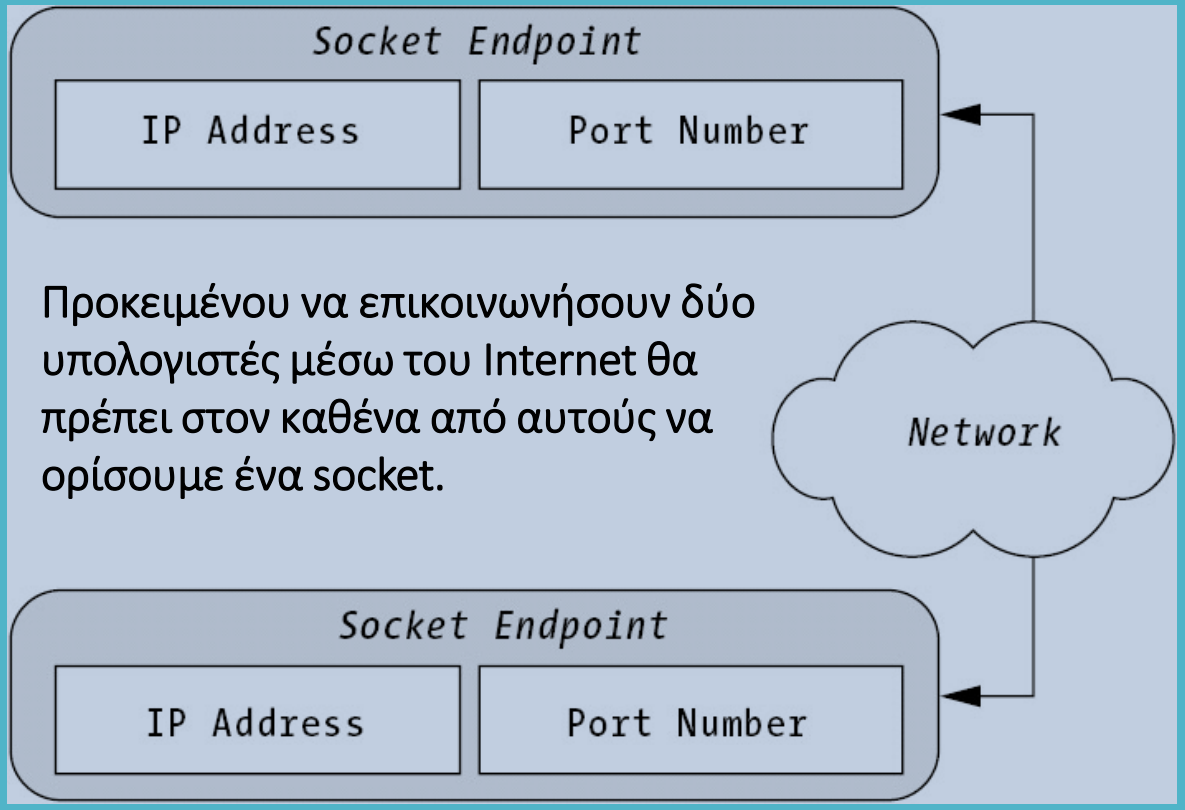
Ένας υπολογιστής του διαδικτύου που προσφέρει κάποια υπηρεσία, προσδιορίζεται πλήρως από το συνδυασμό IP Address : Port Number ο οποίος είναι γνωστός ως TCP / IP socket.



Δομή για TCP / IP socket

```
struct sockaddr_in {
        short           sin_family;
        unsigned short  sin_port;
        struct in_addr  sin_addr;  };
```

sin_family → AF_INET
port → port number
sin_addr → IP Address



Προκειμένου να επικοινωνήσουν δύο υπολογιστές μέσω του Internet θα πρέπει στον καθένα από αυτούς να ορίσουμε ένα socket.

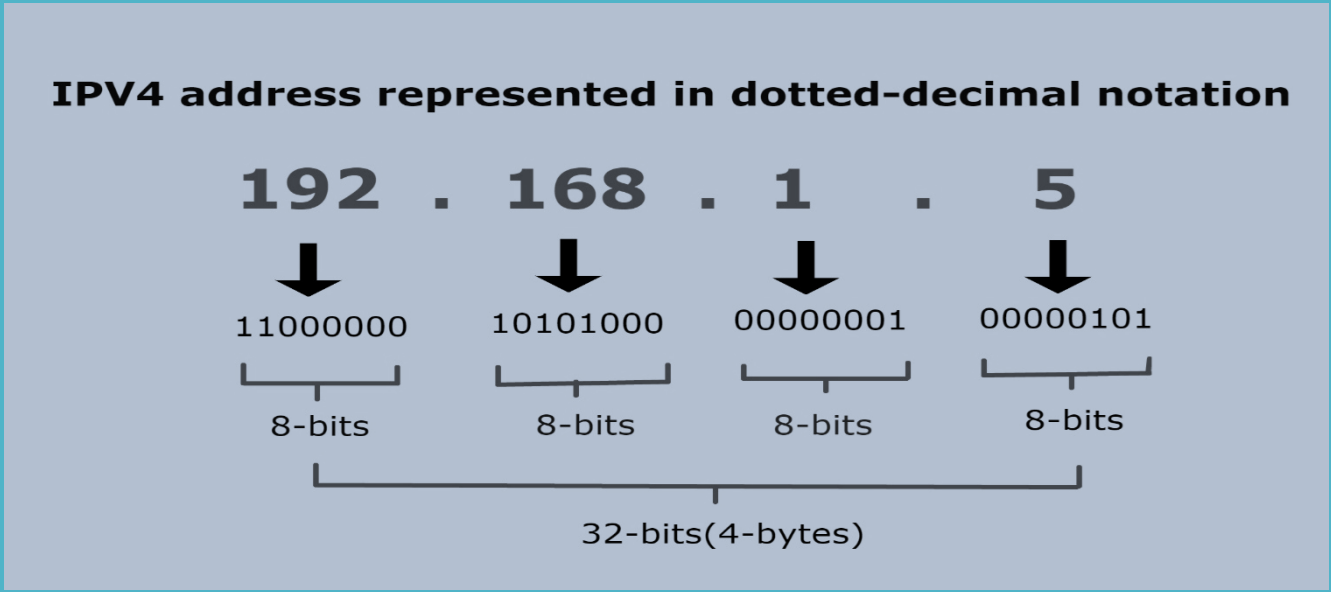
Υποδοχείς

SOCKET PROGRAMMING

Μία διεύθυνση IP μπορεί να εμφανιστεί σε δύο διαφορετικές μορφές και πιο συγκεκριμένα:

Εστιγμένη δεκαδική αναπαράσταση (dotted decimal notation) → η διεύθυνση IP αποτελείται από τέσσερις αριθμούς με τιμές ανάμεσα στο 0 και στο 255 οι οποίοι είναι χωρισμένοι με τελεία.

Δυαδική αναπαράσταση (binary notation) → προκύπτει από τη μετατροπή της παραπάνω αναπαράστασης στο δυαδικό σύστημα. Οι τέσσερις αριθμοί μετατρέπονται στο δυαδικό σύστημα με μήκος 8 bits και στη συνέχεια συνενώνονται. Οι τελείες δεν λαμβάνονται υπόψη.



Υποδοχείς

SOCKET PROGRAMMING

Μετατροπή IP διεύθυνσης από δυαδικό συμβολισμό σε εστιγμένο δεκαδικό συμβολισμό.

char * inet_ntoa (struct in_addr address)

Μετατροπή IP διεύθυνσης από εστιγμένο δεκαδικό συμβολισμό σε δυαδικό συμβολισμό.

unsigned long int inet_addr (const char * ddaddress);

Αν και αυτή η συνάρτηση ήταν ιστορικά η πρώτη που χρησιμοποιήθηκε για αυτό το σκοπό, ωστόσο αποδείχθηκε προβληματική και για το λόγο αυτό αντικαταστάθηκε από την πιο κατάλληλη συνάρτηση

unsigned long int inet_atton
(const char * ddaddress, struct in_addr * address);

Για τη χρήση των παραπάνω συναρτήσεων θα πρέπει να γίνουν include τα header files

<netinet/in.h> και <arpa/inet.h>

Υποδοχείς

SOCKET PROGRAMMING

Παράδειγμα χρήσης TCP / IP socket (Πρωτόκολλο TCP)

SERVER

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>

#define PORT 8080
```

```
int main(int argc, char const *argv[]) {
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Hello from server";
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE); }
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                   &opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE); }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE); }
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE); }
    if ((new_socket = accept (server_fd, (struct sockaddr *)&address,
                              socklen_t*)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE); }
    valread = read (new_socket, buffer, 1024);
    printf("%s\n",buffer );
    return 0; }
```

Ο client αποστέλλει στον server ένα μήνυμα που παραλαμβάνεται και εκτυπώνεται από αυτόν.

Υποδοχείς

SOCKET PROGRAMMING

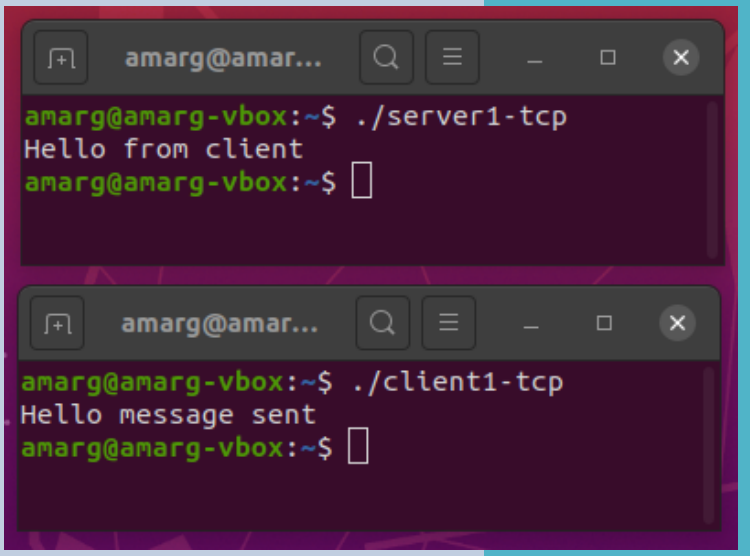
Παράδειγμα χρήσης TCP / IP socket (Πρωτόκολλο TCP)

```c
#include <stdio.h>
#include <sys/socket.h> // AF_INET and SOCK_STREAM
#include <arpa/inet.h>  // storage size of serv_addr
#include <unistd.h>     // getpid, getppid, fork
#include <string.h>     // bzero

#define PORT 8080

int main(int argc, char const *argv[]) {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char *hello = "Hello from client";
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1; }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0) {
        printf("\nInvalid address/ Address not supported \n");
        return -1; }
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1; }
    send(sock , hello , strlen(hello) , 0 );
    printf("Hello message sent\n");
    return 0; }
```

CLIENT



Output

Υποδοχείς

SOCKET PROGRAMMING

Παράδειγμα χρήσης TCP / IP socket (Πρωτόκολλο TCP)

Η εφαρμογή πελάτης συνδέεται σε έναν απομακρυσμένο Web server (port 80) η IP διεύθυνση του οποίου δίδεται από το χρήστη και διαβάζει τις 1000 πρώτες γραμμές του κώδικα HTML της κεντρικής σελίδας τις οποίες και εκτυπώνει.

```c
int main(int argc, char * argv[]) {
    struct sockaddr_in sa;
    int fd_skt;
    char buf[1000];
    ssize_t nread;
    if (argc!=2) {
       printf ("Usage: inetExample xxx.xxx.xxx.xxx\n");
       exit (-1); }
    sa.sin_family = AF_INET;
    sa.sin_port = htons(80);
    sa.sin_addr.s_addr =  inet_addr(argv[1]);
    if ((fd_skt = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
         perror("Message from socket [client] : ");
         exit(-1); }
    if (connect(fd_skt, (struct sockaddr *)&sa, sizeof(sa)) < 0) {
        printf("\nConnection Failed \n");
        return -1; }
    write(fd_skt, REQUEST, strlen(REQUEST));
    nread = read(fd_skt, buf, sizeof(buf));
    (void) write(STDOUT_FILENO, buf, nread);
    close(fd_skt);
    exit(0); }
```

port 80 (http)

```c
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define REQUEST "GET / HTTP/1.0\r\n\r\n"
```

Το πρωτόκολλο HTTP (HyperText Transfer Protocol) απαιτεί τη χρήση της θύρας 80 για την εξυπηρέτηση αιτημάτων σύνδεσης από το διακομιστή.

Υποδοχείς

SOCKET PROGRAMMING

Παράδειγμα χρήσης TCP / IP socket (Πρωτόκολλο TCP)

Αρχικά καλούμε την εντολή ping για να ανακτήσουμε τη διεύθυνση IP του δικτυακού τόπου στον οποίο επιθυμούμε να συνδεθούμε (π.χ. www.google.com)

```
amarg@amarg-vbox:~$ ping www.google.com
PING www.google.com (172.217.19.100) 56(84) bytes of data.
64 bytes from bud02s27-in-f4.1e100.net (172.217.19.100): icmp_seq=1 ttl=111 time=68.8 ms
```

και στη συνέχεια χρησιμοποιούμε αυτή τη διεύθυνση (172.217.19.100) ως όρισμα στην εφαρμογή μας.

```
amarg@amarg-vbox:~$ ./inetExample 172.217.19.100
HTTP/1.0 200 OK
Date: Sun, 18 Oct 2020 08:08:43 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: NID=204=tlcVs7M6fZxpWYf0Hs3wj7YRWWnVBvVL7Zd4PhlEW5TGt_ytDx8dhf6AMv7QyDDM9RNDgDFZ91kR03PKBRf3
E6jcKM37W39p7uX_TqQmtUdfqW6t1z379yf2ohRE5XUxzHxPsEDJ9dqGfSVAuzhPzIZzj4qd5itapKsmW2LaHaU; expires=Mon, 19
-Apr-2021 08:08:43 GMT; path=/; domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="el"><head><meta content="te
xt/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/googleg_sta
ndard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="ecDPX8dwpviLT4NfLtA7Fg==">(f
unction(){window.google={kEI:'C_iLX4uhFI2oa_2KiYAJ',kEXPI:'0,18168,183994,1151585,5662,731,22
```

Υποδοχείς

ΑΝΑΚΤΗΣΗ ΟΝΟΜΑΤΟΣ ΚΕΝΤΡΙΚΟΥ ΥΠΟΛΟΓΙΣΤΗ (HOSTNAME)

Η χρήση διευθύνσεων IP των 32 bits είναι ακατάλληλη για τους ανθρώπους οι οποίοι προτιμούν να χρησιμοποιούν ένα απλό όνομα υπολογιστή (hostname). Η μετατροπή ενός hostname στην διεύθυνση IP του υπολογιστή και αντίστροφα, προσφέρεται από την υπηρεσία DNS (Domain Name Service).

Η ανάκτηση του hostname του τοπικού υπολογιστή γίνεται καλώντας την εντολή hostname ή εκτυπώνοντας την τιμή της μεταβλητής περιβάλλοντος HOSTNAME

```
amarg@amarg-vbox:~$ hostname
amarg-vbox
amarg@amarg-vbox:~$ echo $HOSTNAME
amarg-vbox
```

Ένας τρόπος για να δούμε στην πράξη την υπηρεσία DNS είναι να εκτελέσουμε την εντολή ping η οποία χρησιμοποιεί το πρωτόκολλο ICMP για να ελέγξει εάν ένας υπολογιστής είναι προσπελάσιμος ή όχι.

```
amarg@amarg-vbox:~$ ping www.google.com
PING www.google.com (172.217.169.132) 56(84) bytes of data.
64 bytes from sof02s32-in-f4.1e100.net (172.217.169.132): icmp_seq=1 ttl=111 time=87.2 ms
64 bytes from sof02s32-in-f4.1e100.net (172.217.169.132): icmp_seq=2 ttl=111 time=86.7 ms
64 bytes from sof02s32-in-f4.1e100.net (172.217.169.132): icmp_seq=3 ttl=111 time=89.1 ms
```

Η εντολή ping ανατρέχοντας στον DNS server που είναι δηλωμένος στο σύστημα, αναφέρει πως η διεύθυνση IP του υπολογιστή με hostname www.google.com είναι η 172.217.169.132.

Υποδοχείς

ΑΝΑΚΤΗΣΗ ΟΝΟΜΑΤΟΣ ΚΕΝΤΡΙΚΟΥ ΥΠΟΛΟΓΙΣΤΗ (HOSTNAME)

Σε ένα hostname μπορούν να αντιστοιχούν περισσότερες από μία διευθύνσεις IP και αντίστοιχα, σε μία διεύθυνση IP μπορεί να αντιστοιχούν περισσότερα από ένα hostnames.

```
struct hostent {
   char* h_name;         /* official name of host */
   char** h_aliases;     /* NULL-terminated alias list */
   int h_addrtype        /* address type (AF_INET) */
   int h_length;         /* length of addresses (4B) */
   char** h_addr_list; /* NULL-terminated address list */
};
```

Σε επίπεδο κώδικα η μετατροπή **hostname ↔ IP address** πραγματοποιείται από τις συναρτήσεις

struct hostent * gethostbyname (const char * name);

struct hostent * gethostbyaddr (const char * addr, int len, int type);

εκ των οποίων η πρώτη δέχεται ένα hostname ενώ η δεύτερη μία IP address. Αμφότερες οι συναρτήσεις επιστρέφουν έναν δείκτη σε μία δομή hostent συμπληρωμένη με τις κατάλληλες πληροφορίες.

Υποδοχείς

Αριθμοί θύρας (port numbers)

Κατά τη σύνδεση ενός χρήστη σε έναν υπολογιστή που αντιστοιχεί σε μία συγκεκριμένη διεύθυνση IP, το είδος της εξυπηρέτησης που θα πραγματοποιηθεί, προσδιορίζεται από τον αριθμό θύρας. Γνωστοί αριθμοί θύρας είναι οι 21 (ftp), 22 (ssh), 23 (telnet), 25 (email) και 80 (www).

Στο λειτουργικό σύστημα Linux η αντιστοίχιση των δικτυακών πρωτοκόλλων στους διάφορους αριθμούς θύρας περιλαμβάνεται στο αρχείο /etc/services.

```
amarg@amarg-vbox:~$ cat /etc/services
tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                          # SSH Remote Login Protocol
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
whois           43/tcp          nicname
tacacs          49/tcp                          # Login Host Protocol (TACACS)
tacacs          49/udp
```

Υποδοχείς

Αριθμοί θύρας (port numbers)

Η ανάκτηση των σχετικών πληροφοριών γίνεται με τη συνάρτηση getservbyname του αρχείου netdb.h

struct servent * getservbyname (const char * name, const char * protocol);

όπου name το όνομα της υπηρεσίας για την οποία η εφαρμογή χρειάζεται πληροφορίες και protocol το πρωτόκολλο που θα χρησιμοποιηθεί.

Η getservbyname επιστρέφει έναν δείκτη σε μία δομή servent που ορίζεται ως

```
struct servent  {
        char *s_name;           /* service name */
        char **s_aliases;       /* pointer to alternate service name array */
        int s_port;             /* port number */
        char *s_proto;          /* name of protocol */
};
```

Η κάθε υπηρεσία μπορεί να έχει πολλά ονόματα αλλά έναν και μοναδικό αριθμό θύρας.

Υποδοχείς

Η συνάρτηση getaddrinfo

Εάν ο χρήστης επιθυμεί να συνδεθεί σε ένα υπολογιστή με συγκεκριμένο όνομα (nodename ή hostname) και να χρησιμοποιήσει μία συγκεκριμένη υπηρεσία (service), μπορεί να κατασκευάσει εύκολα μία socket address που θα χρησιμοποιηθεί ως όρισμα στις συναρτήσεις bind και connect καλώντας τη συνάρτηση getaddrinfo ως

```
int getaddrinfo (const char * node, const char * service,
                 const struct addrinfo * hints, struct addrinfo ** res);
```

όπου node του υπολογιστή (που μπορεί να αναζητηθεί σε DNS Server, στο αρχείο /etc/hosts ή να είναι κάποια διεύθυνση IPv4 ή IPv6), service το όνομα της υπηρεσίας και hints ένας δείκτης σε δομή addrinfo που ορίζει κριτήρια για την επιλογή των δομών socket address που επιστρέφονται στη λίστα res.

```
struct addrinfo {
  int          ai_flags;
  int          ai_family;
  int          socktype;
  int          ai_protocol;
  size_t       ai_addrlen;
  struct sockaddr * ai_addr;
  char *       ai_cannonname;
  struct addrinfo * ai_next; }
```

Με την ολοκλήρωση της σύνδεσης απαιτείται αποδέσμευση μνήμης.

```
void freeaddrinfo
     (struct addrinfo * res)
```

Εάν ανακύψει σφάλμα ανακτάται καλώντας τη συνάρτηση

```
const char * gai_strerror
     (int code);
```

Υποδοχείς

Παράδειγμα χρήσης της συνάρτησης getaddrinfo

```
// Source: https://basepath.com/aup/ex/adi_8c-source.html
// Rochkind's Book, page 574

#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <netdb.h>

int main (int argc, char * argv[]) {
    int retVal; char err[50];
    struct addrinfo *infop = NULL, hint;
    memset(&hint, 0, sizeof(hint));
    hint.ai_family = AF_INET;
    hint.ai_socktype = SOCK_STREAM;
    if (argc!=2) {
       printf ("Usage: inetExample xxx.xxx.xxx.xxx\n");
       exit (-1); }
    retVal = getaddrinfo(argv[1], "80", &hint, &infop);
    if (retVal!=0) {
        strcpy (err,gai_strerror(retVal));
        printf ("Error found ==> %s\n", err);
        exit (-1); }
    for ( ; infop != NULL; infop = infop->ai_next) {
        struct sockaddr_in *sa = (struct sockaddr_in *)infop->ai_addr;
        printf("%s port: %d protocol: %d\n", inet_ntoa(sa->sin_addr),
               ntohs(sa->sin_port), infop->ai_protocol); }
    exit(0);}
```

```
amarg@amarg-vbox:~$ ping www.uth.gr
PING zeus.uth.gr (194.177.200.17) 56(84) bytes of data.
64 bytes from zeus.uth.gr (194.177.200.17): icmp_seq=1 ttl=53 time=39.0 ms
amarg@amarg-vbox:~$ ./addrInfo 194.177.200.17
194.177.200.17 port: 80 protocol: 6
amarg@amarg-vbox:~$
```

Ο χρήστης καλεί την ping με μία συμβολική διεύθυνση για να ανακτήσει την πραγματική IP διεύθυνση την οποία στη συνέχεια καταχωρεί ως όρισμα στην εφαρμογή addrInfo.

Υποδοχείς

Ασυνδεσμικές επικοινωνίες (SOCKET_DGRAM)

Δεν χρησιμοποιείται σύνδεση οπότε δεν χρησιμοποιούνται οι listen και accept.

Ο αποστολέας καθορίζει τη διεύθυνση στην οποία επιθυμεί να στείλει ένα πακέτο. Η διεύθυνση αποστολής προστίθεται στο πακέτο το οποίο αποστέλλεται χωρίς να ελέγχεται και να υπάρχει εγγύηση πως αυτό έφτασε σωστά στον παραλήπτη.

Ο παραλήπτης καθορίζει τη διεύθυνση από την οποία επιθυμεί να παραλάβει ή παραλαμβάνει από όλους και ενημερώνεται για τη διεύθυνση του αποστολέα.

Συνήθως δεν χρησιμοποιούνται αρχιτεκτονικές client – server αλλά όλοι οι σταθμοί είναι ομότιμοι μεταξύ τους (peer to peer).

```
ssize_t recvfrom(int sockfd, void *buff, size_t nbyte, int flag,
                 struct sockaddr *from, socklen_t *addrlen);

ssize_t sendto(int sockfd, const void *buff, size_t nbyte, int flag,
               const struct sockaddr *to, socklen_t addrlen);
```

Υποδοχείς

Παράδειγμα ασυνδεσμικής επικοινωνίας

SERVER

```c
int main() {
    int sockfd, len, n;
    char buffer[MAXLINE];
    char *hello = "Hello from server";
    struct sockaddr_in servaddr, cliaddr;
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE); }
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    servaddr.sin_family    = AF_INET; // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
    if (bind(sockfd, (const struct sockaddr *)&servaddr,
             sizeof(servaddr)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE); }
    n = recvfrom (sockfd, (char *)buffer, MAXLINE, MSG_WAITALL,
                  (struct sockaddr *) &cliaddr, &len);
    buffer[n] = '\0';
    printf("Client : %s\n", buffer);
    sendto(sockfd, (const char *)hello, strlen(hello), MSG_CONFIRM,
                   (const struct sockaddr *) &cliaddr, len);
    printf("Hello message sent.\n");
    return 0; }
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT     8080
#define MAXLINE 1024
```

Ανταλλαγή ενός μηνύματος μεταξύ σταθμών οι οποίοι συνήθως είναι ομότιμοι.

Υποδοχείς

Παράδειγμα ασυνδεσμικής επικοινωνίας

```c
int main() {
    int sockfd, len, n;
    char buffer[MAXLINE];
    char *hello = "Hello from client";
    struct sockaddr_in servaddr;
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE); }
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    sendto(sockfd, (const char *)hello, strlen(hello), MSG_CONFIRM,
                   (const struct sockaddr *) &servaddr, sizeof(servaddr));
    printf("Hello message sent.\n");
    n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL,
                         (struct sockaddr *) &servaddr, &len);
    buffer[n] = '\0';
    printf("Server : %s\n", buffer);
    close(sockfd);
    return 0; }
```

CLIENT

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT     8080
#define MAXLINE 1024
```

```
amarg@amarg-vbox:~$ ./client2-udp
Hello message sent.
Server : Hello from server
```

```
amarg@amarg-vbox:~$ ./server2-udp
Client : Hello from client
Hello message sent.
```

Output

Υποδοχείς

Σύνδεση πολλών client σε έναν server

Στην περίπτωση κατά την οποία ένας server δέχεται αιτήσεις εξυπηρέτησης από πολλούς clients, ανακύπτουν ζητήματα που σχετίζονται με τη διαχείριση τερματικών (terminals).

Αυτά σχετίζονται με τα αρχεία συστήματος /dev/tty για τα οποία χρησιμοποιούνται οι γνωστές κλήσεις συστήματος open, close, read και write – ωστόσο μπορούν να χρησιμοποιηθούν και τα γνωστά αρχεία stdin, stdout και stderr.

Κάθε προσπάθεια ανοίγματος ενός τερματικού μπλοκάρεται (αναστέλλει τη λειτουργία της και τίθεται σε αναμονή) μέχρι η συσκευή να συνδεθεί σε κάποιο τερματικό, εκτός εάν χρησιμοποιηθεί το flag O_NONBLOCK.

ΣΥΝΔΕΣΗ ΧΡΗΣΤΗ ΣΤΟ ΣΥΣΤΗΜΑ → Η open επιστρέφει μη μηδενική τιμή στη διεργασία που επιχειρεί τη σύνδεση – στη συνέχεια εκτελείται το login process για την είσοδο του χρήστη και στο τέλος καλείται το login shell.

Η write είναι σχετικά απλή → όσοι χαρακτήρες προωθούνται για έξοδο, μπαίνουν σε μία σειρά προκειμένου να αποσταλούν στο τερματικό.

Η close λειτουργεί ακριβώς όπως στα αρχεία.

Υποδοχείς

Σύνδεση πολλών client σε έναν server

Ωστόσο, η read λειτουργεί διαφορετικά! ΔΕΝ επιστρέφει, παρά μόνο όταν ολοκληρωθεί η γραμμή, δηλαδή όταν ο χρήστης πατήσει το ENTER και αυτό, επειδή ο χρήστης μπορεί να αναθεωρήσει (←) . Επίσης, επιστρέφει μία μόνο γραμμή κάθε φορά.

Με άλλα λόγια, η συνάρτηση μπλοκάρει ... ωστόσο, στην περίπτωση κατά την οποία συνδέονται στον server πολλά τερματικά (π.χ μετρητικές διατάξεις) είναι επιθυμητή η κατάργηση του μπλοκαρίσματος.

Για ποιο λόγο συμβαίνει αυτό?

Διότι μπορούμε να περιμένουμε άσκοπα από ένα τερματικό να στείλει δεδομένα χωρίς αυτό να έχει κάτι να στείλει και ταυτόχρονα να αγνοούμε άλλα τερματικά που έχουν δεδομένα για αποστολή και τα οποία δεν μπορούν να στείλουν.

Αναζητούμε μία συνάρτηση η οποία να παραμένει αδρανής μέχρι να εμφανιστεί διαθέσιμος χαρακτήρας εισόδου για οποιονδήποτε περιγραφέα αρχείου. Αυτή η συνάρτηση είναι η

```
int select (int nfds, fd_set * readfds, fd_set * writefds,
                      fd_set * errorfds, struct timeval * timeout);
```

και επιτρέπει την ταυτόχρονη παρακολούθηση πολλών περιγραφέων αρχείων αναμένοντας μέχρι ένας ή περισσότερους από αυτούς να γίνει έτοιμος για την πραγματοποίηση κάποιας λειτουργίας I/O.

Υποδοχείς

Σύνδεση πολλών client σε έναν server

Τα ορίσματα readfds, writefds και errorfds είναι δείκτες σε σύνολα περιγραφέων αρχείων που ορίζουν ποιοι περιγραφείς αρχείων θα παρακολουθεί η συνάρτηση.

Το όρισμα nfds συνήθως είναι ίσο με max {fds}+1 όπου max {fds} το μέγιστο πλήθος περιγραφέων που παρακολουθεί η select.

Το όρισμα timeout είναι δείκτης σε μία δομή timeval που ορίζεται ως

```
struct timeval{
        int tv_sec;   /* seconds*/
        int tv_usec; /* microseconds*/
        };
```

και καθορίζει το χρονικό διάστημα που θα περιμένει η select προκειμένου να συμβεί κάτι. Εάν η τιμή αυτής της παραμέτρου είναι NULL η select θα διακοπεί μέχρι την εκδήλωση κάποιου συμβάντος.

Η select επιστρέφει το συνολικό πλήθος των στοιχείων που εξετάζονται στα τρία σύνολα περιγραφέων αρχείου, την τιμή 0 εάν έχει λήξει ο χρόνος για την κλήση και την τιμή -1 εάν έχει εκδηλωθεί σφάλμα.

Υποδοχείς

Σύνδεση πολλών client σε έναν server

Η select διαχειρίζεται τα σύνολα περιγραφέων αρχείων μέσω των επόμενων μακροεντολών.

FD_ZERO (fd_set * fds)

Αρχικοποιεί το σύνολο fd_set απομακρύνοντας από αυτό όλους τους περιγραφείς αρχείων.

FD_SET (int fd, fd_set * fds)

Προσθέτει τον περιγραφέα αρχείου fd στο σύνολο fds. Για παράδειγμα, μετά την κλήση της FD_ZERO εάν θέλουμε να εντάξουμε στο σύνολο fds τους περιγραφείς fd1 και fd2 θα εκτελέσουμε τις εντολές

```
FD_SET (fd1, &set);          FD_SET (df2, &set);
```

FD_CLR (int fd, fd_set * fds)

Απομακρύνει τον περιγραφέα αρχείου fd από το σύνολο fds.

FD_ISSET (int fd, fd_set * fds)

Ελέγχει εάν ο περιγραφέας αρχείου fd ανήκει στο σύνολο fds.

Υποδοχείς

Σύνδεση πολλών client σε έναν server

Έστω πως μία διεργασία θέλει να διαβάσει δεδομένα από 3 sockets, από μία διάταξη SCSI και από ένα αρχείο και έστω πως οι τιμές των περιγραφέων αρχείου είναι οι 3, 7, 10, 12 και 67. Στην περίπτωση αυτή, το όρισμα readfds έχει τιμή

```
00010001.00101000:00000000.00000000:00000000.00000000:00000000.00000000   00010000.00000000
   ^   ^   ^ ^                                                                 ^
   0   0   1 1                                                                 6
   3   7   0 2                                                                 7
```

FD_SET (fd1, &set); (όπου fd1=03)
FD_SET (fd2, &set); (όπου fd2=07)
FD_SET (fd3, &set); (όπου fd3=10)
FD_SET (fd4, &set); (όπου fd4=12)
FD_SET (fd5, &set); (όπου fd5=67)

Η select τροποποιεί τα σύνολα που δέχεται και προκειμένου να ελέγξουμε εάν ένας περιγραφέας αρχείου είναι ενεργός ή όχι, καλούμε τη μακροεντολή ISSET.

Μία ανταγωνιστική παραλλαγή της select είναι η poll που αντί για μάσκες bits ενημερώνει ολόκληρες δομές , ενώ εκτός από επιλογή πραγματοποιεί και σταθμοσκόμηση (polling).

Υποδοχείς

Σύνδεση πολλών client σε έναν server

Η συνάρτηση run_server μέσα από ένα infinite loop καλεί συνεχώς την accept για να συλλάβει αιτήματα σύνδεσης από τέσσερις διεργασίες – πελάτες, χρησιμοποιώντας τη select. Στη συνέχεια επικοινωνεί με τον καθέναν από τους πελάτες ανταλλάσσοντας με αυτόν ένα μήνυμα.

Οι πελάτες αποτελούν θυγατρικές διεργασίες που δημιουργούνται με κλήση της fork μέσα στην αντίστοιχη συνάρτηση run_client.

```c
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/un.h>

#define SOCKETNAME "MySocket"
```

SERVER

```c
static int run_server(struct sockaddr_un *sap) {
    int fd_skt, fd_client, fd_hwm = 0, fd;
    char buf[100];
    fd_set set, read_set;
    ssize_t nread;
    printf ("Server pid is %d\n", getpid());
    if ((fd_skt = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n"); return -1; }
    if (bind(fd_skt, (struct sockaddr *)sap, sizeof(*sap)) < 0) {
        perror("Bind"); exit(EXIT_FAILURE); }
    if (listen(fd_skt, SOMAXCONN)<0) {
        perror("Listen"); exit(EXIT_FAILURE); }
    if (fd_skt > fd_hwm) fd_hwm = fd_skt;
    FD_ZERO(&set);
    FD_SET(fd_skt, &set);
    while (1) {
        read_set = set;
        if (select(fd_hwm+1, &read_set, NULL, NULL, NULL)==-1) {
            perror("Select"); exit(EXIT_FAILURE); }
        for (fd = 0; fd <= fd_hwm; fd++)
            if (FD_ISSET(fd, &read_set)) {
                if (fd == fd_skt) {
                    if ((fd_client = accept(fd_skt, NULL, 0))<0) {
                        perror("Select"); exit(EXIT_FAILURE); }
                    FD_SET(fd_client, &set);
                    if (fd_client > fd_hwm) fd_hwm = fd_client; }
                else {
                    if ((nread = read(fd, buf, sizeof(buf)))<0) {
                        perror("Read"); exit(EXIT_FAILURE); }
                    if (nread == 0) {
                        FD_CLR(fd, &set);
                        if (fd == fd_hwm) fd_hwm--;
                        close(fd); }
                    else {
                        printf("Server got \"%s\"\n", buf);
                        if (write(fd, "Goodbye!", 9)==-1) {
                            perror("Write"); exit(EXIT_FAILURE); }}}}}
    close(fd_skt);
    close(fd_client);
    printf ("Server terminates\n");
    return 1; }
```

Υποδοχείς

Σύνδεση πολλών client σε έναν server

CLIENT

```
static int run_client(struct sockaddr_un *sap) {
    if (fork() == 0) {
        int fd_skt;
        char buf[100];
        if ((fd_skt = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {
             printf("\n Socket creation error \n"); return -1; }
        while (connect(fd_skt, (struct sockaddr *)sap, sizeof(*sap)) == -1) {
            if (errno == ENOENT) {
                sleep(1);
                continue; }
            else {
             perror("Message from connect [client]");
             exit(-2); }}
        snprintf (buf, sizeof(buf), "Hello from %ld!", (long)getpid());
        if (write (fd_skt, buf, strlen(buf)+1) < 0) {
            perror("Write"); exit(EXIT_FAILURE); }
        if ((read(fd_skt, buf, sizeof(buf))) < 0) {
            perror("Read"); exit(EXIT_FAILURE); }
        printf("Client %d got %s\n", getpid(), buf);
        close(fd_skt);
        printf ("Client %d terminates\n", getpid());
        exit(EXIT_SUCCESS); }
      return 1; }
```

Output

```
Server pid is 3082
Server got "Hello from 3086!"
Server got "Hello from 3085!"
Server got "Hello from 3087!"
Client 3086 got Goodbye!
Client 3086 terminates
Client 3087 got Goodbye!
Client 3085 got Goodbye!
Server got "Hello from 3084!"
Client 3087 terminates
Client 3085 terminates
Client 3084 got Goodbye!
Client 3084 terminates
```

Main

```
int main(void) {
    struct sockaddr_un sa;
    int nclient;
    (void)unlink(SOCKETNAME);
    strcpy(sa.sun_path, SOCKETNAME);
    sa.sun_family = AF_UNIX;
    for (nclient = 1; nclient <= 4; nclient++)
         run_client(&sa);
    run_server(&sa);
    exit(EXIT_SUCCESS); }
```

Η συνάρτηση main δημιουργεί τέσσερις πελάτες και στη συνέχεια δημιουργεί έναν διακομιστή για να τους εξυπηρετήσει.